SN Computer Science (2020) 1:162
https://doi.org/10.1007/s42979-020-00167-2

ORIGINAL RESEARCH

A New Role of Pseudorandom Number as Inductive Inference for Solving Classic Coin-Weighing Puzzles

Yoshiyasu Takefuji[1] · Ren Yamada[1]

Received: 23 December 2019 / Accepted: 15 April 2020

Abstract
This paper demonstrates that constrained pseudorandom number can solve classic coin-weighing puzzles faster than human-devised algorithms. Deep learning and ensemble machine learning lie in inductive methods while automated reasoning implemented in deductive computer languages is based on deductive methods. In the inductive methods, intelligence is inferred by pseudorandom number for creating the sophisticated decision trees. In machine learning, the goal is to learn a function that minimizes an error or one that maximizes reward over punishment. This paper shows a new challenge that every solution candidate is solely generated by the constrained pseudorandom number, while a simple deductive rule is applied for selecting solutions among the generated candidates. In this paper, the computation performance of the proposed method was measured by comparing with the existing open source codes by solving classic 12-coin and 24-coin puzzles, respectively. The proposed method can be used for solving quantitative group testing in biology and medicine.

Keywords Inductive methods · Pseudorandom number · Classic coin-weighing puzzles

Introduction

Artificial intelligence has been used for solving intractable problems. Machine learning [1] has been outperforming human world champions in games including Chess [1], Go [2], Shogi [3], and Quiz bowl questions [4, 5], respectively, in the year of 2017. 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.

There are three types of artificial intelligence methods: inductive, deductive, and hybrid 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.

Neural Turing Machines (NTM) [6] and Differentiable Neural Computers (DNC) [7] originated by DeepMind Inc, are the representative studies of the inductive method. NTM is an artificial intelligence technology that implements external memory to improve long short-terms memory (LSTM), and DNC is an improved version that enhances memory operations of NTM. However, since both methods (NTM/DNC) are only statistical and inductive method, the problem still remains that data constrains the results statistically and inductively. Their system is complicated by implementing external memory on the originally complex LSTM. On the other hand, the proposed method uses a new hybrid method. In the proposed hybrid method, solution candidates are generated inductively and solely by constrained pseudorandom number, and solutions among the generated solution candidates are selected by a simple deductive rule. Because of the hybrid method, the final results are bound by the simple deductive rule. The proposed method solves the problem surprisingly simply without using complicated mechanisms including LSTM with external memory in NTM/DNC.

Several hybrid methods have been recently proposed where they are based on fusing inductive and deductive

✉ Yoshiyasu Takefuji
takefuji@keio.jp

1 Graduate School of Media and Governance, Keio University, Fujisawa, Japan

Published online: 12 May 2020

method [8, 9]. However, the proposed hybrid method is much simpler than the existing hybrid methods.

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 [10], Otter [11], and Z3 [12] 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 [13–18], we realize that intelligence is truly inferred by pseudorandom number in artificial intelligence. This paper demonstrated a new challenge for solving classic coin-weighing puzzles by the hybrid method with pseudorandom number for generating solution candidates and a simple deductive rule for selecting solutions among the generated candidates. The proposed claim in this paper is that the proposed pseudorandom number approach may outperform expert-devised algorithms in constrained optimization problems.

In order to justify the proposed claim, the computation time is compared with that of the existing algorithm (publicly available execution codes) by solving classic 12-coin and 24-coin puzzles, respectively. The machine with Intel(R) Core(TM) i7-6600U CPU @ 2.6 GHz and 16 GB memory was used for all performance evaluations in this paper.

The proposed hybrid method is simply composed of two components: an inductive inference component generating constrained solution candidates solely by using pseudorandom number and a deductive rule for selecting solutions among generated solution candidates.

What is the Classic 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.

Solving the coin-weighing problem is also called experimental design. The goal of the experimental design is to devise a test design that identifies the infected individuals with the least number of tests. This is also called quantitative group testing. Quantitative group testing has been used in biology and medicine.

Inductive Inference 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 methods. 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 methods) 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.

Performance Comparisons with Deductive Methods in 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 computation 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.

(A) 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
```

(B) 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]),0,1)
        b_l = If(Distinct(i[4:8]+[c_p]),0,1)
        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])
        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()
```

(C) 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
real    0m0.241s
$ time ./odd.pl 24 4
real    0m58.129s
```

(D) The proposed pseudorandom method

In the proposed pseudorandom method, pseudorandom number is solely used for generating solution candidates. 12coins.py and 24coins.py in Python 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 solely 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 unique coins (no duplicated 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 unique 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 1. The proposed pseudorandom method outperforms

Table 1 Computation time comparison for 12-coin and 24-coin puzzles

	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

the existing open source programs in 12-coin-3-weighing and 24-coin-4-weighing puzzles, respectively.

This paper shows how to convert classic coin-weighing puzzles into nonlinear encoding problems. Inductive inference can be inferred by constrained pseudorandom numbers in machine learning as we mentioned. Without human intelligence, classic coin-weighing puzzles can be solved by the hybrid pseudorandom method using an inductive inference and a simple deductive inference which is composed of 39 lines of source code in Python language. The goal means that we must determine the simple deductive rule (1) how many times weighing a balance scale and (2) how many coins per set are chosen for balancing. In 12-coin-3-weighing puzzle, 8 coins are pseudorandomly selected by three times. In 24-coin-3-weighing puzzle, 16 coins are pseudorandomly selected by three times.

The proposed nonlinear encoding program is composed of two components: an inductive inference component determining how many coins per experiment are pseudorandomly selected and a deduction rule component whether the generated experiments are correct or not. In the classic 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 of the classic coin-weighing puzzles 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: $3 \times 3 \times 3 = 27$. Using 27 possible ways, we must distinguish a fake coin among 12 coins.

The experimental design can be fixed as follows: an experiment set composed of 8 coins which are pseudorandomly and uniquely selected from 12 coins and it is repeated three times. In other words, in three experiments, the total number of 24 coins ($= 8$ coins $\times 3$ times) must be inductively selected. The pseudorandom number plays a key role in generating three experiments where 8 coins are pseudorandomly 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 inductive inference can alleviate us to create a set of complex logic or rules for finding valid solutions. In the conventional methods, the algorithms must be created by algorithm experts. In the proposed method, all we need to do is to define experimental design and create constraints for coding the requirements in order to find solutions.

Pseudorandomly generated three experiments must distinguish 24 possible states (12 coins, heavier or lighter per coin) until the requirements are all satisfied. In other words, the validation goal in encoding the problem is to generate three satisfactory experiments where 24 possible states can be uniquely 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 uniquely distinguished by three experiments (yellow colored program).

In general, the program may be 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.

In the program, the following array of 12×24 indicates 24 states. For example, [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] means that the first coin is heavier than others. [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1.] shows that 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.

```
[ 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. ]
[-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 pseudorandomly 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.

You can access to the following web service: https://nrich.maths.org/5796 for validating 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 s per successful solution on laptop with Intel i7-6600U 2.6 GHz. It takes 0.04 s 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:['>=<', '<>=', '<<>', '=<<', '>>>', '==>', '>>=', '<=<', '<==', '=>=', '=<>', '><<', '===']
```

Features of the Proposed Method are Summarized

1. The deductive method is used.

Many studies of algorithmic learning attempt to learn inductively and statistically using externally given data. According to the inductive method, interpolation by an approximate function can be used to make a reasonable estimation for regions with data, but it is difficult to say that the estimation is appropriate for regions without data because it is extrapolated from the approximate function. This study, on the other hand, introduces a deductive method based on the findings of the symbolic and logical system. According to the deductive method, since the object of learning is the action itself, it is possible to act to some extent more moderately in areas where there is no data than in the inductive method.

2. Hybrid use of deductive and inductive methods.

Many previous studies have used either deductive or inductive methods, but this study is a hybrid of the two. In this way, not only do they complement the strengths and weaknesses of each method, but they also function so that the deductive method expresses intrinsic motivation and the inductive method expresses extrinsic motivation, respectively.

3. Simple and fast model architecture

Where the architecture of previous studies has become too complex, this study proposes a novel method that is implemented by a hybrid method that combines inductive methods of statistical and knowledge systems and deductive methods of symbolic and logical systems. This method is simple as the source code is presented in the paper, and the architecture is fast.

4. Usage of constrained pseudorandom numbers

While uniformly random numbers are often used in previous studies to ensure the completeness of state information, this paper uses constrained pseudorandom numbers to provide pragmatic benefits for fast computation.

Conclusion

This paper shows a new role of pseudorandom number for solving coin-weighing puzzles. The pseudorandom number plays a key role in inductive intelligence in artificial intelligence systems. The proposed hybrid method using constrained pseudorandom number with a simple deductive rule outperforms the existing open source codes in Perl, Prolog, and Z3, respectively. In the proposed hybrid method, pseudorandom number is solely used for generating solutions candidates efficiently and solutions among the generated candidates are selected by the simple deductive rule. The deductive rule is 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). Therefore, three experiments give us $27 = 3 \times 3 \times 3$ possible ways in coding. In the 12-coin-3-weighing puzzle, detecting a fake (heavier or lighter) coin among 12 coins must distinguish unique $24 (= 12 \times 2)$ states. The proposed hybrid method is a general purpose scheme for solving optimization problems. The pseudorandom number generation can be combined with the stronger constraints for reducing searching space in order to further minimize the computation time in the future.

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

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Géron A. Hands-on machine learning with scikit-learn and TensorFlow. Sebastopol: O'Reilly; 2020.
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. Graves A et al. Neural turing machines. https://arxiv.org/pdf/1410.5401.pdf. 2014.
7. Graves A, et al. Hybrid computing using a neural network with dynamic external memory. Nature. 2016;538(7626):471.
8. Li T et al. Augmenting neural networks with first-order logic. https://arxiv.org/pdf/1906.06298.pdf. 2019.
9. Weber L et al. NLProlog: reasoning with weak unification for question answering in natural language. https://arxiv.org/pdf/1906.06187.pdf. 2019.
10. http://www.swi-prolog.org/.
11. https://www.mcs.anl.gov/research/projects/AR/otter/.
12. https://rise4fun.com/z3/tutorial.
13. Takefuji Y, Lee KC. A near-optimum parallel planarization algorithm. Science. 1989;245:1221–3.
14. Takefuji Y. Neural network parallel computing. Berlin: Springer; 1992.
15. Pao YH, Takefuji Y. Functional-link net computing: theory, system architecture and functionalities. IEEE Comput. 1992;25(5):76–9.
16. Amartur SC, Piraino D, Takefuji Y. Optimization neural networks for the segmentation of magnetic resonance images. IEEE Trans Med Imaging. 1992;11(2):215–20.
17. Yamada I, Tamaki R, Shindo H, Takefuji Y. Studio Ousia's quiz bowl question answering system. In: Escalera S, Weimer M, editors. The NIPS'17 competition: building intelligent systems., The Springer series on challenges in machine learningCham: Springer; 2018.
18. Funabiki N, Takefuji Y. A neural network parallel algorithm for channel assignment problems in cellular radio networks. IEEE Trans Veh Technol. 1992;41(4):430–7.