JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 425-439 (2021)
DOI: 10.6688/JISE.202103_37(2).0010

Seed Optimization Framework on Draughts

CHING-NUNG LIN[1], FABIEN LETOUZEY, OLIVIER TEYTAUD[2] AND SHI-JIM YEN[1]
[1]*Department of Computer Science and Information Engineering*
National Dong Hwa University
Hwalien, 974 Taiwan
[2]*TAO, INRIA, France*
E-mail: 810221001@gms.ndhu.edu.tw; fabien letouzey@hotmail.com;
olivier.teytaud@inria.fr; sjyen@mail.ndhu.edu.tw

Seed optimization has been successfully tested on many games such as Go, Domineering, Breakthrough, among others. Fixed seeds can outperform random seeds by selecting locally optimal seeds as different playing policies. In this article seed optimization has been tested for the Draughts program Scan. We provide a framework which can optimize a draughts program for competition. It does not affect the original program structure, so it improves the strength with no modifying algorithm and no penalty when executing. With the new Best Promise Seed framework, the win rate can be improved by replacing the random seeds with some pretested locally optimal seeds. The optimized program won the championship in the Computer Olympiad in 2015 and 2016. It shows that self learning methodology improves the strength of Scan against other competing programs. In addition, better locally optimal seed(s) may be discovered with a longer learning time, so further strength improvement is possible. All current draughts programs and other different game programs might gain benefit from this framework.

Keywords: draughts, seed optimization, BestSeed, policy optimization, machine learning

1. INTRODUCTION

Computer international draughts is a popular game in the Computer Olympiad; there are many draughts programs competing each year. The program Scan with the seed optimization strategy won the championship in the Computer Olympiad in 2015 (12 programs) and 2016 (10 programs). Computer Olympiad is the most important computer game tournament in the world. Scan achieves a level of play competitive with that of human professionals. This paper implements the Best Promise Seed framework which learns by self-play games. As a result, 34 Elo points are gained, which is a big success since the program already was able to play at a strength similar to that of human champions. The training started two months before the 2015 ICGA Computer Olympiad. This research is the first result showing improvements in competitive play to show that seed optimization can, by self-play, further improve a program strategy that is already state of the art. Also, the learning improves the win rate against other programs.

Random numbers are often used in board game programs. As a result, the program may not produce the same move in the same situation every time. Some moves may be better than others. For example, Scan uses random numbers to generate hash keys used in a transposition table, and it affects how a move is chosen. The BestSeed [1] method considers that "fixed suboptimal" seeds can play good moves. However, it is impossible to

Received October 14, 2018; revised December 15, 2018 & January 29, 2019; accepted February 21, 2019.
Communicated by Chao-Lin Liu.

find the best seed(s) in an infinite space. Seed optimization refers to finding locally optimal seed(s) in a finite seed space and using them in place of random seeds. The performance after seed optimization plays at least equal to or better than playing with random seeds (by cross-validation and the central limit theorem). This research applies the enhanced BestSeed method to the game of draughts and shares the positive results. The fixed seed methods have outperformed the random seeds in many experiences [1-5]. Similarly, seed optimization is a method to evaluate seeds' performance by constructing a self-play matrix and then confirming that it is locally optimal by cross-validating. If no locally optimal seed is found, a larger self-play matrix can be built and verified. Finally, select the locally optimal seed(s) to replace the random seed(s). On the other hand, using fixed seed(s) to optimize a program is similar to constructing an opening book. It can be confirmed that this opening book plays better than random playing, but since the number of games required for confirmation is unknown, scalability becomes a practical bottleneck.

This paper contains five noteworthy innovations. First, our best promise seed framework improves the game performance more than did the previous BestSeed method. Second, we show that our framework succeeded in a real world application. In the 2015 and 2016 Computer Olympiad it never lost any game to the other competing programs. In addition, since our framework focuses only on adjusting a single random seed, it does not require a major revision of the program. For this reason, it might be efficiently generalized to other programs because most programs use at least one random seed to compete with different opponents in a game competition. A few tips are given to encourage readers to try it in their own programs. Third, this is the first research to optimize an already very strong program; always a difficult challenge. It does so without modifying any algorithm, and without requiring additional resources. Fourth, our framework is practical because it can scale up in a distribution system more easily than other methods. Fifth, on the basis of 4 million games, it shows that the game of draughts is slightly biased in favor of the player moving second, and the draw rate is very high when two players are equally strong.

The rest of the paper is divided as follows: Section 2 introduces the latest technology of computer draughts, the draughts program Scan and seed optimization. Section 3 shows in detail how to tune a program in this framework with the use of a self-play matrix, cross-validation and the best promise seed selection. Section 4 presents the experimental results. Section 5 gives a conclusion.

2. RELATED WORKS

2.1 Computer Draughts

Draughts is EXPTIME-complete [6]. International (10×10) draughts cannot be weakly solved by current technology. Although draughts programs had already appeared by the 1970s [7], recent ones are still making progress. The current top programs are, in no particular order: Dragon Draughts by Michel Grimminck [8], Kingsrow International by Ed Gilbert [9], and Scan by Fabien Letouzey [10]. Experts generally believe that the best programs and human champions are of similar strength. The high draw rate (> 90%) might be delaying the computer domination that we are witnessing in other games.

Similarly, to chess engines, draughts programs combine alpha beta search with a fast

evaluation function.

Commonly used enhancements are: Principal Variation Search (PVS), quiescence search for captures, transposition tables, history heuristics, forward pruning, and parallel search. Furthermore, implementations rely heavily on bitboards for move generation and position evaluation. Null move pruning, however, does not work in draughts because it is a game of Zugzwang [11].

Tactics are ubiquitous in the game, but easily covered by short searches since captures [12]. Therefore, the focus is on positional play. Programmers used to spend a lot of time translating positional knowledge from strategy books or personal experience into actual code. Recently there has been an increased interest in machine learning techniques to improve evaluation. Although convolutional neural networks (CNNs) have good accuracy, they are also computationally intensive. The current top programs now all use "patterns" [13, 14] which are much faster.

Most programs include an opening book, which is often constructed automatically using best first search variants such as dropout expansion [15, 16]. They also access endgame table bases built using retrograde analysis [17]. To help com pression, only win/loss/draw information is used during search. All positions with up to six or eight pieces (depending on the program) are covered, and some programs have partial information (such as "draw or better") on selected positions with more than eight pieces.

A good presentation of computer draughts techniques can be found in [18], although positional patterns are not covered. Many technical terms are described on [19]. Up to date information can be found on the draughts programmers' forum [20].

2.2 Scan

Scan is a new program, first released in 2015 but already state of the art. It won the Computer Olympiad in 2015 and 2016. Scan is open source and freely available [10].

Scan is a traditional game program as described in the previous subsection 2.1. Its design is rather close to Othello programs, combining a fast and accurate evaluation with heavy forward pruning. The pruning heuristics are a simplified variant of ProbCut [13], and late move reductions [21]. More details can be found on the programmers' forum [22].

Scan's evaluation function is centered around patterns [13, 14]. The board is divided into overlapping regions (the patterns) and every possible instance (a configuration) in a region gets a score, computed using machine learning. The final evaluation is the sum of local scores.

Scan uses 4×4 patterns (see Fig. 1 for an example). Since only the dark squares are used in draughts, each pattern actually covers eight squares. For a given pattern (group of squares), the white/empty/black square contents are combined into a base three integer, one digit per square, which is used to index an array containing the corresponding score [14].

The pattern configuration scores were learned as a preprocessing step using logistic regression. A large database of self-play games was used as the training set (supervised learning). Note that no learning is taking place during game play. The impact of random numbers in Scan is on Zobrist hash keys [23], which affect the transposition table. This Zobrist hash keys are generated once at the initialization. There are total 201 (2×2×50+1) random number generated because there are 2 players, 2 piece types and 50 legal squares.

The extra one is to present the current playing color. The transposition table implementtation is the same as all modern programs [20]: There are 64 bits whose first 20 bit is for the index and the next 32 bit is for "lock" (states) which is to detect different positions in the same index. The remaining stores the depth, score, boundary status and the best move. There are 4 buckets in each entry. For the replacement strategies, the one with smallest search depth is replaced by the new one. If the depth is the same, older entries are always picked when it is available. The time for each entry is increased by one before every new search (the game move).

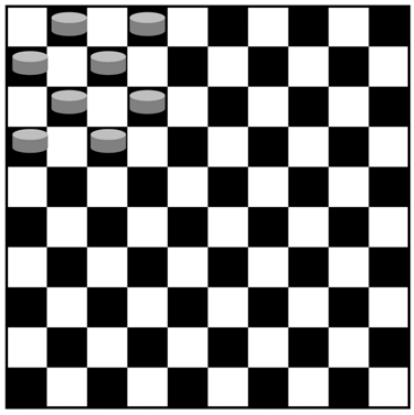

Fig. 1. A 4×4 pattern on a draughts board.

2.3 Seeds Optimization

Seed optimization is suitable both for two player games such as Chess and for multiplayer games such as Bridge or Mahjong. It is an offline learning process that treats different seeds as different policies. Instead of randomly choosing policies, learning how to efficiently implement policies resulted in better performance. There are several different policy selection methods such as the BestSeed and Nash Equilibrium (NE) distribution. The study [2] showed that choosing policies with an NE probability distribution outperforms the random seeds in the game of Go. However, the BestSeed method is superior in most cases. The partially observable game Phantom Go [4] has been tested as well. Because the seed is fixed, the policy remains the same, which is similar to playing a game with the same opening moves. However, the games will be different because the opponent move is changed in certain depth with constraints as time, search space, random seeds, among others. Lately, there are more games such as Bridge, Chess, Havannah, Batoo and Variants that have been tested and gained solid improvement [3, 5].

Since seeds optimization is successful in many games, how to efficiently reduce matrix density becomes an important issue because constructing a full matrix is computationally intensive. The research [1] provides a method to construct part of the full matrix. The rectangular matrix which is constructed comprises a fraction such as 1/10, 2/10, ... of the full matrix. The performance of the partial matrix in the games of Domineering, AtariGo and Breakthrough does not degrade too much in comparison to the full matrix, when the

size of the partial matrix is big enough. Also, there are Exp3 and Bandit [1] methods to construct a partial matrix without sacrificing significant performance.

However, the above tests are verified with cross-validation exclusively. No test is performed against another program. Only in the game of Go, a GNU Go Monte Carlo with different seed policies against a traditional GNU Go is tested which is considered as two different programs; the one with fixed seeds shows a slightly better win rate. The random seeds are used in many parts: Pattern weights, Zobrist hashing and Move selection. In the Monte Carlo version, it is used on random simulation additionally. In addition, the Bridge program "Wbridge5" [5] won the World Computer Bridge Championship in 2016 with the similar BestSeed method. First, it builds a self-play 40×40 matrix. Because it is a partial information game, each element in this matrix is decided by 64 games. Then, it uses cross-validation to confirm that there are local optimal seeds. Finally, it plays games uniformly selecting from the best k% seeds. Other methods might be used to find the best seeds, but most lack parallel scalability. For example, the use of a Genetic algorithm is difficult to scale up to thousands of processes in a distribution system.

3. BEST PROMISE SEED OPTIMIZATION FRAMEWORK

The major difference between the best promise framework and previous BestSeed methods lies in selecting the best seed(s) from multiple overlapping submatrices in a matrix instead of the seed(s) with the most winning games in this matrix. The best promise seed framework is defined so as to enhance the BestSeed method for a more concrete result.

There are three steps to implement the best promise seed framework in ap plications: self-play matrix, cross-validation and seed selection. First, construct a matrix with your application by self-play games which uses the same program but different seeds. Second, cross-validate the result in this matrix to ensure that learning makes progress. Finally, select those seeds that demonstrate the best performance on this program. In practice, for playing in a competition with few games, using the best promise framework or the BestSeed method is preferred.

3.1 Self-play Matrix

A matrix is built from the result of self-play games with fixed seeds. In this matrix, the row is seeds playing first and the column is seeds playing second. This matrix is used to find the locally optimal seed(s).

For example, program A with seed equal to 1 plays first against program A with seed equal to 2. If playing first (seed 1) wins, the M [1, 2] is set to 1. If playing second (seed 2) wins, the M [1, 2] is set to 0. Similarly, M [2, 9] is set to 1⁄2 if the game is draw between program A with seed equal to 2 playing first and program A with seed equal to 9 playing second. The result is set to 1, 1⁄2, 0 when playing first wins, draws, and loses respectively. A matrix of three or more dimensions can be constructed for games of more than two players. For partial information games such as Chinese Dark Chess, the result in a matrix can be calculated by averaging many different games (different initial boards) with the same seed setting.

3.2 Cross-validation

The cross validation is to ensure there is at least one locally optimal seed in the current matrix. Divide this matrix for training and validating data. If the learning result is greater than 50% when validating it, it means playing with the fixed seed(s) outperforms playing with the random seed(s) in this matrix. In Algorithm 1, randomly choose i seeds as training data until 95% of the matrix size N (i starting from size equal to 2). Then the training process begins. There are a total of S samples in each iteration. In each sample, one best seed can be calculated. Then this seed is validated with the remaining seeds. If the win rate is greater than 50%, it has succeeded. Then, the final win rate for each iteration is the average of the win rate of total samples. The sampling size affects the standard deviation. In addition, if the win rate increases when the training size increases, the learning is progressing.

Algorithm 1: Cross-validation

Require: Matrix M of size N+1, Sample size S
Ensure: exist array *aFirst*, *aFirstR*, *aSecond*, *aSecondR*, Matrix MR of size N+1
  for i = 1 to N **do**
    for j = 1 to N **do**
      $MR_{ij} = 1 - M_{ij}$
    end for
  end for
  for i = 2 to $N \times 95\%$ **do**
    ValidateFirstWin = 0, *ValidateSecondWin* = 0
    for j = 1 to S **do**
      aFirst ← randomly choose i numbers from 1 to N
      aFirstR ← $\{k|1 \le k \le N, k \notin aFirst\}$
      aSecond ← randomly choose i numbers from 1 to N
      aSecondR ← $\{k|1 \le k \le N, k \notin aSecond\}$
      TrainFirstBestSeed ← arg max($\sum_{M[aFirst][aSecond]}$)
      TrainSecondBestSeed ← arg max($\sum_{M[aFirst][aSecond]}$)
      ValidateFirstWin = ValidateFirstWin + $\sum_{M[TrainFirstBestSecond][aSecondR]}$
      ValidateSecondWin = ValidateSecondWin + $\sum_{MR[aFirstR][TrainSecondBestSeed]}$
    end for
    Win Rate = 50%×ValidateFirstWin/(S×i)+50%×ValidateSecondWin/(S×i)
  end for

Table 1. Left is the 5×5 matrix *E* for playing first and right is the 5×5 matrix ER for playing second.

seed	1	2	3	4	5
1	1	1/2	0	1	0
2	1	1/2	0	1	1/2
3	1/2	1/2	0	1	1/2
4	1/2	1/2	0	1	0
5	0	1/2	0	1	0

seed	1	2	3	4	5
1	0	1/2	1	0	0
2	0	1/2	1	0	1/2
3	1/2	1/2	1	0	1/2
4	1/2	1/2	1	0	0
5	1	1/2	1	0	0

For example, suppose there is a matrix E and a matrix ER as in Table 1. The matrix E is the result of playing first with seeds 1 to 5; the matrix ER is a complementary matrix of E which is helpful for future calculation. The row is seeds of the current player which is playing first in the matrix E and playing second in the matrix ER. The column is seeds of the opposite player which is playing second in the matrix E and playing first in the matrix ER. In this sample, there are two seeds randomly chosen: seed 1 and seed 4 for playing first; seed 1 and seed 3 for the playing second. TrainFirstBestSeed can be learned by calculating $E[1, 1] + E[1, 3] = 1 + 0$ and $E[4, 1] + E[4, 3] = 1/2 + 0$, so the seed 1 is the best seed learned for playing first. In the same way, TrainSecondBestSeed is calculated as $ER[1, 1] + ER[1, 4] = 0 + 0$, $ER[3, 1] + ER[3, 4] = 1/2 + 0$, so the seed 3 is the best seed learned for playing second. For validation, calculate the win rate with the trained best seed and the seeds that are not in the training data. These results are $E[1, 2] + E[1, 4] + E[1, 5] = 1/2 + 1 + 1$ for playing first and $ER[3, 2] + ER[3, 3] + ER[3, 5] = 1/2+1+ 1/2$ for playing second. Finally, the win rate is $50\% * 2.5/3 + 50\% * 2/3 = 75\%$. This learning is an improvement because using this pair of seeds achieves a better win rate than 50%.

3.1 Seeds Selection

If the win rate of cross validation is increasing when the learning size increases, the seeds selection can be processed. Otherwise, a larger matrix is required to be constructed (go back Subsection 3.1). Since there is at least one locally optimal seed in the matrix, the better seed(s) can be used to replace the random seed(s). In previous Best Seed method [1], choose the seed(s) which has the most winning games. However, in Algorithm 2, the best promise seed selection chooses the seed(s) which has the highest "promise score". The promise score gives a higher value to the seeds in a larger submatrix because the score in the larger submatrix is more statistically robust than the one in the smaller submatrix. Here, the score is defined as the sum of winning points from the winning games (1 for a win, 1/2 for a draw, and 0 for a loss). As a result, averaging all the scores from the small to large submatrices in the same matrix gives the promise score. To calculate the promise score, the sample interval I and top rank K% must be defined. Then, there are N/I square submatrices formed by taking the first N/I, $N/I * 2$, ..., N rows and columns in this matrix. In ith submatrix ($i = 1, 2, ..., N/I$), the best ranked K% of seeds are collected out of the first $N/I * i$ seeds. The promise score for each seed is the number of times it was collected divided by the times it was considered. Finally, the best promise seed(s) is the seed(s) with the highest promise score when playing first and playing second. It is possible that there is more than one best promise seed. When this occurs, more tests can be done with certain seeds since the number of seeds are greatly reduced.

Algorithm 2: The best promise seed selection

Require: Matrix M of size $N + 1$, Sample interval I, Top rank K
Ensure: exist array *PromiseScoreFirst*, *PromiseScoreSecond*, *ScoreFirst*, *ScoreSecond*, Matrix MR of size $N + 1$

```
for i = 1 to N do
    for j = 1 to N do
        MR_ij = 1 − M_ij
    end for
```

end for
submatrixNumber ← $\frac{N}{I}$
for i = 1 to submatrixNumber **do**
  indexBound ← $I \times i$
  indexNumber ← $\{k|1 \le k \le indexBound\}$
  for j = 1 to indexBound do
    $ScoreFirst_{ij} = M[j][\text{indexNumber}]$
    $ScoreSecond_{ij} = MR[j][\text{indexNumber}]$
  end for
end for
for i = 1 to N **do**
  $PromiseScoreFirst_i \leftarrow 0$
  $PromiseScoreSecond_i \leftarrow 0$
end for
for i = 1 to submatrixNumber **do**
  indexBound ← $I \times i$
  indexNumber ← $\{k|1 \le k \le indexBound\}$
  indexScore ← $\{k|1 \le k \le \text{ScoreFirst}[i][indexNumber]\}$
  rankSeed ← choose the K% highest score seed(s) from indexScore
  for $j \in$ rankSeed **do**
    $PromiseScoreFirst_j = PromiseScoreFirst_j + 1$
  end for
  indexScore ← $\{k|1 \le k \le ScoreSecond[i][indexNumber]\}$
  rankSeed ← choose the K% highest score seed(s) from indexScore
  for $j \in$ rankSeed **do**
    $PromiseScoreSecond_j = PromiseScoreSecond_j + 1$
  end for
end for
for i = 1 to N **do**
  proportion ← $\left\lfloor submatrixNumber - \frac{i-1}{I} \right\rfloor$
  $PromiseScoreFirst_i = \frac{PromiseScoreFirst_i}{proportion}$
  $PromiseScoreFirst_i = \frac{PromiseScoreSecond_i}{proportion}$
end for
The best promise seed ← arg max(*PromiseScoreFirst*,*PromiseScoreSecond*)

To illustrate Algorithm 2, we use a mini example with ten seeds. We construct a 10×10 matrix M (see Table 2). The sample interval I is set to 5 and top rank K is set to 20. Therefore, there is two square submatrices formed by taking the first 5 and 10 rows and columns in M. In the 5×5 submatrix. The score of seed 1 is M [1, 1] + M [1, 2] + M [1, 3] + M [1, 4] + M [1, 5] = 3.5, similarly, the score for seed 2, 3, 4, and 5 can be calculated as 3.0, 2.5, 3.0, 1.5. Because K is 20, the top 20% rank seeds are chosen in each sub-matrix. Thus, the best seed in this submatrix is seed 1. Similarly, we calculate the 10×10 submatrix, the Top 2 ranked seeds are chosen which are seed 7 whose score is 9.0 and seed 1 whose score is 8.0.

Table 2. 100 games result of a self-play 10×10 matrix.

seed	1	2	3	4	5	6	7	8	9	10
1	1	1/2	0	1	1	1	1	1	1/2	1
2	1	1/2	0	1	1/2	0	1/2	0	0	0
3	1/2	1/2	0	1	1/2	1	1	1	1	1
4	1/2	1/2	0	1	1	0	0	1/2	0	0
5	0	1/2	0	1/2	1/2	1	1	1	1/2	1/2
6	1/2	1/2	0	1	1/2	1	1	0	0	1
7	1	0	1	1	1	1	1	1	1	1
8	0	1/2	0	1	1/2	1	1	1/2	1	1
9	0	0	0	0	0	0	1	0	1	1
10	0	1	0	1	1	1	1/2	1/2	1/2	1

Now, we can calculate the promise score of each seed. Since seed 1 was considered in two submatrices and was chosen both times, the promise score is 2/2. Seed 7 was considered once and was selected, so its Promise Score is 1/1. Similarly, we calculate the Promise Score when playing second as same as previously. Finally, the best promise seed are 1, 7 and 9 whose Promise Score is 1. This selection is efficient, there are only three seeds required to be tested instead of ten seeds after the selection.

For another example, in our experiments, we construct a 2000×2000 matrix. When I is set to 100 and *K* is set to 1, there would be 20 square submatrices formed by taking the first 100, 200, ..., 2000 rows and columns in this matrix. In each submatrix, the bestranked 1% of seeds are collected. In the 100×100 submatrix, the seed with the highest score is selected; In the 1000×1000 submatrix, the ten seeds with the highest scores are collected. Then we can find PromiseScore by averaging of the overall occurrence frequency for each seed in all submatrices and find the best promised seed(s).

4. EXPERIMENTAL RESULTS

All the self-play games are computed on INRIA Tompouce Cluster. A matrix is constructed using the result of games, the index is the seed number. Each game whose computational resource limits is equal to what the Gigabyte P34 notebook (i74710HQ) is able to play in the 2015 ICGA Computer Olympiad draughts tournament time setting which is 30 minutes for each side. All win rate is tested with the strong opensource draughts program Moby Dam [24] as the baseline. Moby Dam is the only other strong open source program available currently, the default opening book and the 6piece endgame database are used. All tests are run on the same machine (Gigabyte P34) as used in the Computer Olympiad. Scan rarely loses to this opponent (< 0.01%), so most results are wins or draws. Each test is composed of 2000 games as half playing first and half playing second; the standard error is around 1.118%.

In Subsection 4.1, we compare the different seed methods with the self-play 300×300 matrix used in the 2015 Computer Olympiad. In Subsection 4.2, When the time setting is decreased, the selected seeds might not result in as much improvement. In Subsection 4.3, we compare the performance of different seed methods in a larger self-play 2000×2000 matrix. We use the result of cross validation to show the effect of increasing the matrix size. Then, we describe how to select the seeds with BestSeed and the best promise seed

methods. Finally, we test the performance among BestSeed, the best promise seed and the random seeds. In Subsection 4.4, since there are so many games, a statistical analysis will be presented.

4.1 Best Promise Seed Method Performance Against Other Seed Methods

In the 2015 and 2016 Computer Olympiads, we find seed 12 of the best promise seed framework based on the self-play 300×300 matrix available at that time. After constructing the matrix, we use cross validation to test the effect. Fig. 2 shows that the win rate rises when the matrix size increases. There exist some local optimal seeds.

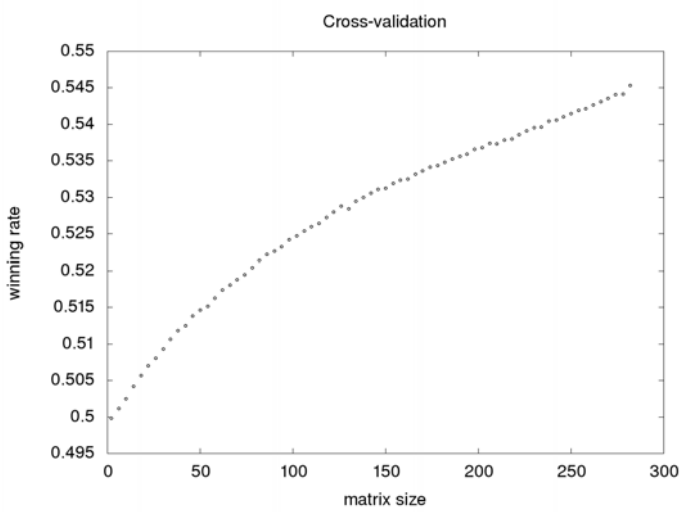

Fig. 2. Result of cross validation of the 300×300 matrix. The sample size (S) is set to 10000, so the standard deviation is small

In Table 3, the seed 12 and seed 178 are selected as the best promise seeds because their promise scores are 1 (3/3 and 2/2) when *I* is set to 100 and *K* is set to 1. We choose seed 12 to play in the tournament. In comparison, BestSeed method chooses the seed(s) that has the most winning games, which is seed 12 for playing first and seed 178 for playing second.

Table 3. Table of the result of top 1% Seeds when the interval is set to 100 at a self-play 300×300 matrix.

	3	2	1		
1	12	44	12	288	**3**
2	241	12	57	249	**2**
3	240	59	178	178	**1**
		1	2	3	

In the upper left triangle, the row is the rank of winning scores and the column is the interval number. Seed 12 is at row 1 and column 1 which indicates the highest score in the 100×100 submatrix. Also, it is at row 2 and column 2 which indicates the second highest score in the 200×200 submatrix.

Table 4. Winning rate of different seed methods against Moby Dam. Each test is composed of 2000 games as half playing first and half playing second; the standard error is around 1.118%.

Methods	Winning Rate
Best promised seed	60.8%
BestSeed	59.3%
Random seeds	55.95%

In Table 4, the best promise seed method performs slightly better than Best Seed method. Both fixed seed methods perform better than using random seeds.

4.2 Different Time Setting

The seed optimization is specific for a fixed time setting. The result of using Scan with the seed equal to 12 in the tournament time setting (20 minutes) is statistically better than random seeds (See Table 5); the time setting is less than 30 minutes because operators must play on a real board and use a clock in the tournament. However, others are slightly better but under two standard deviations in the 95% confidence interval.

Table 5. Result of different time settings. Each test is composed of 2000 games as half playing first and half playing second; the standard error is around 1.118%.

Time setting(minutes)	20	10	5	1
Seed 12	60.8	59.4%	61.64%	68.49%
Random	**55.95%**	57.2%	59.5%	66.3%

4.3 The Effect of Increasing the Matrix Size

After the Computer Olympiad, a larger self-play matrix is constructed. We use cross validation to test the effect when increasing the matrix size. After constructing the 2000× 2000 matrix, the cross validation win rate grows slowly. Even though the draw rate is so high, the cross validation confirms that increasing the matrix size works. Fig. 3 shows that the win rate rises when the matrix size increases in the 2000×2000 matrix.

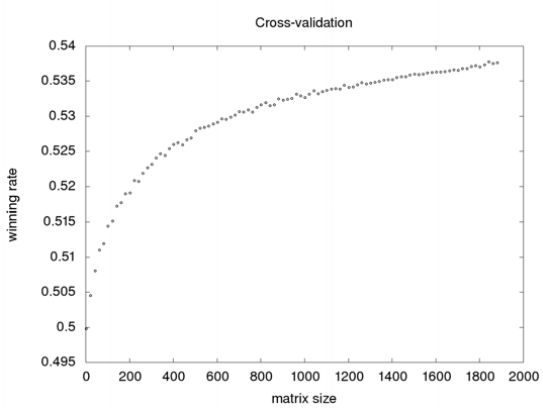

Fig. 3. Result of cross validation of the 2000×2000 matrix. The sample size (S) is set to 10000, so the standard deviation is small.

Table 6 describes those seeds with the highest scores found when the interval is set to 100 ($I = 100$) and the top rank is set to 1% ($K = 1$) in the 2000×2000 matrix. The upper left triangle is the playing first matrix, and the lower right triangle is the playing second matrix. The column indices refer to the interval numbers. For example, the index 17 refers to the 1700×1700 submatrix. The bold text means the rank of the seeds. For example, seed 826 is the top 2 scoring seed when playing first in the 1900×1900 submatrix and seed 775 is the top 8 scoring seed when playing second in the 900×900 submatrix.

Using the BestSeed method, seed 826 at the top left corner and seed 1924 at the bottom right corner are selected because they are both seeds with the highest winning scores. On the other hand, with the best promise seed method, those top 1% score seeds in each submatrix are identified as good seeds. There are a total of 16 seeds (178, 314, 412, 456, 664, 909, 924, 1082, 1173, 1269, 1296, 1376, 1425, 1440, 1924, 1938) whose promise scores are 1. However, seed 826 is not selected because it does not appear in the 900×900, 1000×1000, …, 1500×1500 submatrices. It does not perform well against smaller seeds.

Table 7 shows the win rate of the seeds selected by the best promise seed method. Each seed is different for playing first and playing second. For the best promise seeds, only 16 seeds are required to be verified which reduces the searching space enormously compared to checking seeds from 1 to 2000 for 2000 games each.

Table 6. Table of the result of top 1% Seeds when the interval is set to 100. In the upper left triangle, the lower row is the rank of winning scores and the column is the interval number.

Seed	178	314	412	456	664	909	924	1082
First	59.2%	59.3%	57.0%	**61.4%**	57.1%	56.9%	56.7%	54.9%
Second	56.4%	60.5%	60.5%	55.4%	59.2%	60.6%	59.0%	58.8%

Table 7. Table of the win rate against Moby Dam. Each test is composed of 2000 games as half playing first and half playing second; the standard error is around 1.118%.

Seed	1173	1269	1296	1376	1425	1440	1924	1938
First	58.9%	58.7%	57.5%	55.3%	55.3%	55.2%	54.8%	57.6%
Second	60.8%	56.7%	59.5%	55.2%	61.3%	59.3%	**63.1%**	61.0%

Table 8. Winning rate of different seed methods against Moby Dam. Each test is composed of 2000 games as half playing first and half playing second; the standard error is around 1.118%.

Methods	Winning Rate
Best promised seed	62.25%
BestSeed	59.85%
Random seeds	55.95%

Table 8 shows that both fixed seed methods perform better than using random seeds. With the best promise seed method, selecting seed 456 as playing first and seed 1924 as playing second achieves a 62.25% win rate. With BestSeed method, selecting seed 826 as playing first and seed 1924 as playing second results in a 59.85% win rate. Both methods have better win rate than 55.95% with random seeds.

4.4 Draw Rate and Playing Second Benefit

Table 9 shows that in draughts the draw rate is very high especially when two programs are close. These games are played with fixed seeds. For example, Scan with seed 1 plays against Scan with seed 100 in the same 2015 Computer Olympiad time setting. The seed is set from 1 to 2000 for each side, so there are 4 million games played. It seems that playing second gains some benefit.

Table 9. Table of the result of 4 million scan self-play games with seeds 1 to 2000.

Draw	playing first wins	playing second wins
97.812675%	0.988675%	1.19865%

5. CONCLUSION

This is the first real world case showing that fixed seeds can tune a strong program to become better. This framework can be constructed by self-play. Also, it improves the overall strength against another program which was not used during training. This framework does not affect the original program structure, so it improves the strength with no penalty when executing. However, the matrix construction is computationally intensive. How to reduce the size of the matrix element without decreasing performance remains an interesting research topic.

In the future, we will test the existing 2000×2000 matrix with the partial matrix, Exp3 and Bandit methods to check the trade off between density and performance. Also, there are more real world applications tested such as games of Chess and Chinese Dark Chess (partial information).

ACKNOWLEDGMENT

The authors would like to thank anonymous referees for their valuable comments in improving the overall quality of this paper, and the INRIA internships programme, France. This work was supported in part by the Ministry of Science and Technology of Taiwan under contracts 1082634F259001 through Pervasive Artificial Intelligence Research (PAIR) Labs, Taiwan.

REFERENCES

1. J. Liu, O. Teytaud, and T. Cazenave, "Fast seed learning algorithms for games," in *Proceedings of International Conference on Computers and Games*, 2016, pp. 58-70.
2. D. L. St-Pierre and O. Teytaud, "The nash and the bandit approaches for adversarial portfolios," in *Proceedings of IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1-7.
3. D. L. StPierre, J. Hoock, J. Liu, F. Teytaud, and O. Teytaud, "Automatically reinforcing a game AI," *CoRR*, Vol. abs/1607.08100, 2016.
4. T. Cazenave, J. Liu, F. Teytaud, and O. Teytaud, "Learning opening books in partially

observable games: Using random seeds in phantom go," in *Proceedings of IEEE Conference on Computational Intelligence and Games*, 2016, pp. 1-7.
5. V. Ventos, Y. Costel, O. Teytaud, and S. T. Ventos, "Boosting a bridge artificial intelligence," in *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, 2017, pp. 1-19.
6. A. S. Fraenkel, M. R. Garey, D. S. Johnson, T. Schaefer, and Y. Yesha, "The complexity of checkers on an *n*×*n* board," in *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, 1978, pp. 55-64.
7. L. Nagels, "Draughts programs," http://home.kpn.nl/nagel580/Compdam/CompHist.htm, 2011.
8. M. Grimminck, "Dragon," http://mdgsoft.home.xs4all.nl/draughts/index.html, 1996.
9. E. G. Kingsrow, http://edgilbert.org/Checkers/KingsRow.htm, 2006.
10. F. Letouzey, "Scan 2.0.," https://hjetten.home.xs4all.nl/scan/scan.html, 2015.
11. M. Fierz, "Basics of strategy game programming: part II," http://www.fierz.ch/strategy2.htm, 2002.
12. CPWteam, "Extensions," http://chessprogramming.org/Extensions, 2016.
13. M. Buro, "Experiments with multi prob cut and a new high quality evaluation function for othello," *Games in AI Research*, 1997, pp. 77-96.
14. M. Buro, "Improving heuristic minimax search by supervised learning," *Artificial Intelligence*, Vol. 134, 2002, pp. 85-99.
15. T. R. Lincke, "Strategies for the automatic construction of opening books," in *Revised Papers from the 2nd International Conference on Computers and Games*, 2002, pp. 74-86.
16. T. R. Lincke, "Exploring the computational limits of large exhaustive search problems," Ph.D. Dissertation, Technische Wissenschaften ETH Zürich, Nr. 14701, 2002.
17. R. Lake, J. Schaeffer, and P. Lu, "Solving large retrograde analysis problems using a network of workstations," Department of Computing Science, University of Alberta, 1993.
18. J. J. van Horssen, "Schwarzman vs. maximus: A man machine match in international draughts," *ICGA Journal*, Vol. 35, 2002, pp. 106-119.
19. CPWteam, "Chess programming WIKI," http://chessprogramming.org/, 2016.
20. FMJD, "World draughts forum," http://fmjd.org/bb3/viewforum.php?f=53, 2015.
21. CPWteam, "Late move reductions," http://chessprogramming.org/Late Move Reductions, 2016.
22. FMJD, "Scan technical description," http://laatste.info/bb3/viewtopic.php?p=112682#p112682, 2015.
23. A. L. Zobrist, "A new hashing method with application for game playing," Technical Report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1969.
24. H. Jetten, https://hjetten.home.xs4all.nl/mobydam/mobydam.html, *Moby Dam*, 2015.

Ching-Nung Lin received his M.S. and Ph.D. degrees in Computer Science and Information Engineering in 2013 and 2019 respectively at National Dong Hwa University, Hualien, Taiwan. He was evolved in The French National Institute for Computer Science and Applied Mathematics "Internships" programme in University of Paris-Sud, France, 2015. His research interests include artificial intelligence and computer board games.

Fabien Letouzey is a Software Developer. He received his B.S. and M.S. degrees in Computer Science in 1994 and 1998 respectively at the University of Lille I, France. His interests include board games, machine learning, and programming languages. He is the author of the strong chess programs Fruit, Chess-64 and Senpai, the Othello programs Turtle, Snail, Piloth, Decapus (10×10) and Octopus (8×8), the latter Gold medal winners at the 20th Computer Olympiad, Leiden 2017, the UCI-Winboard adapter Polyglot, and the International Draughts programs Toy, and Scan, which surprised the Draughts scene in winning the 18th Computer Olympiad, Leiden 2015, and defended its title in 2016 and 2017 respectively.

Olivier Teytaud is a Research Scientist at Inria. He has been working in numerical optimization in many realworld contexts-scheduling in power systems, in water management, hyperparameter optimization for computer vision and natural language processing, parameter optimization in reinforcement learning. He is currently maintainer of an open source derivative free optimization platform 'Nevergrad', containing various flavors of evolution strategies, Bayesian optimization, sequential quadratic programming, Cobyla, Nelder-Mead, differential evolution, particle swarm optimization, and a platform of testbeds including games, reinforcement learning, hyperparameter tuning and real-world engineering problems.

Shi-Jim Yen have received his Ph.D. in Computer Science and Information Engineering from National Taiwan University, Taiwan. He is a Distinguished Professor of Department of Computer Science and Information Engineering and the Director of the AI Centre in National Dong Hwa University, Taiwan. He specialized in AI, machine learning and computer games. He is an amateur 6 dan Go player.