Learning opening books in partially observable games: using random seeds in Phantom Go

Tristan Cazenave[1], Jialin Liu[2,3], Fabien Teytaud[4], and Olivier Teytaud[2]

[1]Lamsade, Univ. Paris Dauphine, Paris, France
Email: `cazenave@lamsade.dauphine.fr`
[2]Inria, CNRS UMR 8623, Univ. Paris-Sud, Gif-sur-Yvette, France
Email: `{lastname.firstname}@inria.fr`
[3]Department of Computer Science, Univ. of Essex, Colchester, UK
Email: `jialin.liu@essex.ac.uk`
[4]Lisic, Univ. Littoral, France, Calais, France
Email: `fabien.teytaud@lisic.univ-littoral.fr`

***Abstract*—Many artificial intelligences (AIs) are randomized. One can be lucky or unlucky with the random seed; we quantify this effect and show that, maybe contrarily to intuition, this is far from being negligible. Then, we apply two different existing algorithms for selecting good seeds and good probability distributions over seeds. This mainly leads to learning an opening book. We apply this to Phantom Go, which, as all phantom games, is hard for opening book learning. We improve the winning rate from 50% to 70% in 5x5 against the same AI, and from approximately 0% to 40% in 5x5, 7x7 and 9x9 against a stronger (learning) opponent.**

I. Introduction

A. *Offline learning in games*

Offline learning in games can be e.g. endgame table building [1], opening book construction by self-play [2], or parameter estimation [3]. We propose the use of Random-Seed-portfolios (Section III-A) for offline learning in games for which randomized AIs perform well. This approach will essentially, though not only and not explicitly, learn at the level of the opening book. Learning opening books is particularly hard in partially observable games, due to the difficult belief state estimation; therefore, this recent approach by random seeds is particularly suitable in this case.

The random seeds approach has already been proposed for the game of Go, but the present paper is, to the best of our knowledge, the first application to *partially observable games*, and our resulting algorithm outperforms by far the original algorithm, which is at the current top level in Phantom Go, including its traditional board sizes. This is mainly obtained through opening book learning - which is a hard task in partially observable games.

B. *Randomized artificial intelligences*

1) Why randomizing AIs.: There are games in which optimal policies are randomized and, beyond that, in many cases the state of the art is made of randomized algorithms, in particular since the advent of Monte Carlo Tree Search [4], [5]. Randomized AIs are also required when the AI should be robust to "overfitting" by an opponent - i.e. when we do not want an opponent to be able to learn, by repeated games, a simple winning strategy. A deterministic AI is certainly not suitable in such a case, e.g. for playing on a server or for the pleasure/education of a human opponent. Still, we point out that our approach makes sense in terms of pure performance against the baseline algorithms.

2) The original Monte Carlo approach.: The Monte Carlo approach in games goes back to [6]. The basic idea is to evaluate a position thanks to random simulations. The value at a state s is obtained by averaging the result of hundreds of games played randomly from this state s. This is compliant with partially observable games by randomly sampling the hidden parts of the state. With ad hoc randomization, this approach is the state of the art in Phantom Go.

3) Improvements of the original Monte Carlo approach.: The original Monte Carlo method for games has been vastly improved [7], [8]. For fully observable games it was outperformed by Monte Carlo Tree Search [4], which adds a tree search to the Monte Carlo evaluation principle. For fully observable puzzles (one player games), nested Monte Carlo often outperforms Monte Carlo [9], [10]. In partially observable games with large hidden state, Monte Carlo remains at the top of game programming [11], [12].

C. *Boosting randomized artificial intelligences and learning opening books*

Randomized AIs can be seen as random samplers of deterministic policies. A random seed is randomly drawn, and then a deterministic AI, depending on this seed, is applied. The choice of the random seed is usually considered of negligible importance. However, a recent work [13] has shown that random seeds have an impact, and that the bias inherent to the use of a given randomized AI (which has an implicit probability distribution on random seeds) can be significantly reduced by analyzing the impact of random seeds. We here extend this work to a more challenging case, namely Phantom Go.

Section II describes our testbed for experiments. Section III describes our approach for boosting random seeds. Section IV presents experimental results.

II. Phantom Go

The game of Phantom Go is a two-player game with hidden information. It consists in playing Go without seeing the other player's moves. Each player does not see the board of the other player. In addition, there is a reference board that is managed by a referee and that the players do not see either. On each player's turn, the player proposes a move to the referee. If the move is legal on the reference board, it is played on the reference board and it is the other player's turn. If the move is illegal on the reference board, the referee tells the player that the move is illegal and the player is asked to play another move. The referee is in charge of maintaining the reference board and of telling illegal moves to the players. The game is over when the two players pass.

Monte Carlo methods have been used in Phantom Go since 2005. The resulting program plays at the level of strong human Go players. Monte Carlo Phantom Go was one of the early success of Monte Carlo methods in Go and related games. The principle of Monte Carlo Phantom Go is to randomly choose a "determinization" (i.e. a filling of the unknown parts of the state space) consistent with the previous illegal moves before each playout. For each possible move, the move is simulated, followed by a determinization and a random playout. Thousands of such determinizations and playout sequences are played for each move and the move with the highest resulting mean is played.

This simple method has defeated more elaborate methods using Monte Carlo Tree Search in the former computer Olympiads. Using a parallelization of the algorithm on a cluster our program won five gold medals and one silver medal during the last six computer Olympiads. When winning the silver medal, the program lost to another program using the same method. The program also played three strong Go players in exhibition matches during the 2011 European Go Congress and won all of its three games.

III. Random seeds and their boosting

A. Seeds in games

We consider a randomized artificial intelligence (AI), equipped with random seeds. Our experiments will be performed on a Monte Carlo approach for Phantom Go, though the method is generic and could be applied to any randomized algorithm such as those cited in Section I-B.

We can decide the seed - and when the seed is fixed, the AI becomes deterministic. The original (randomized) algorithm can be seen as a probability distribution over these deterministic AIs.

Algorithm 1 The BestSeed algorithm for boosting a randomized AI. There is a parameter K; K greater leads to better performance but slower computations. The resulting AI is deterministic, but it can be made stochastic by random permutations of the 8 symmetries of the board.

```
Require: $K$, and a randomized AI.
1: for $i \in \{1, \ldots, K\}$ do
2:     for $j \in \{1, \ldots, K\}$ do
3:         Play a game between
             • an AI playing with seed $i$ as Black;
             • an AI playing with seed $j$ as White.
4:         $M_{i,j} \leftarrow 1$ if Black wins, 0 otherwise
5:     end for
6: end for
7: $i_0 \leftarrow \text{argmax}_{i \in \{1,\ldots,K\}} \sum_{j=1}^{K} M_{i,j}$
8: $j_0 \leftarrow \text{argmin}_{j \in \{1,\ldots,K\}} \sum_{i=1}^{K} M_{i,j}$
9: return The (deterministic) AI using seed $i_0$ when playing
   Black and $j_0$ when playing White.
```

A Random-Seed-portfolio (RS-portfolio) consists in optimizing the probability distribution on random seeds. Such an algorithm has been proposed in [13]. We recall below the two algorithms they propose, namely Nash and BestSeed. In both cases, the learning of the probability distribution is based on the construction of a $K \times K$ binary matrix M, where $M_{i,j} = 1$ if Black with random seed i wins against White with random seed j, and $M_{i,j} = 0$ otherwise. This matrix is the learning set; for validating our approach in terms of performance against the original randomized algorithm, we use random seeds which are not in this matrix, and distributed as in the original randomized algorithm.

B. Strategies for choosing seeds

We describe here two methods for choosing a probability distribution on rows $i \in \{1, 2, \ldots, K\}$ and a probability distribution on columns $j \in \{1, 2, \ldots, K\}$. These probability distributions are then used as better probability distributions on random seeds at the beginning of later games.

1) BestSeed approach.: BestSeed is quite simple; the probability distribution for Black has mass 1 on some i such that $\sum_{j \in \{1,\ldots,K\}} M_{i,j}$ is maximal. We randomly break ties. For White, we have probability 1 for some j such that $\sum_{i \in \{1,\ldots,K\}} M_{i,j}$ is minimum. The BestSeed approach is described in Algorithm 1. This method is quite simple, and works because

$$\lim_{K \to \infty} \frac{1}{K} \sum_{j=1}^{K} M_{i,j}$$

$$(resp.\ \lim_{K \to \infty} \frac{1}{K} \sum_{j=1}^{K} M_{j,i})$$

is far from being a constant when i varies.

2) Nash approach.: This section describes the Nash approach. It is more complicated than the BestSeed approach, but it is harder to "exploit", as detailed in the experimental section. First, we introduce constant-sum matrix games, and then we explain how we use them for building portfolios of random seeds.

a) Constant-sum matrix games: We consider constant-sum matrix games; by normalizing matrices, we work without loss of generality on games such that the sum of the rewards for player 1 and for player 2 is one. Consider the following game, parametrized by a $K \times K$ matrix M. Black plays i. White is not informed of Black's choice, and plays j. The reward for Black is $M_{i,j}$ and the reward for White is $1 - M_{i,j}$.

It is known [14], [15] that there exists at least one Nash equilibrium (x, y) such that if Black plays i with probability x_i and White plays j with probability y_j, then neither of the players can improve its expected reward by changing unilaterally his policy. More formally:

$$\exists(x,y),\ \forall(x',y'),\ x'^t My \leq x^t My \leq x^t My',$$

where x, y, x' and y' are non-negative vectors summing to one. Moreover, the value $v = x^t My$ is unique - but the pair (x, y) is not necessarily unique.

It is possible to compute x and y in polynomial time, using linear programming [16]. Some faster methods provide approximate results in sublinear time [17], [18]. Importantly, these fast approximation algorithms are mathematically proved and do not require all the matrix to be available - only $O(K \log(K)/\epsilon)$ elements in the matrix have to be computed for a fixed precision $\epsilon > 0$ on the Nash equilibrium.

b) Nash portfolio of random seeds: Consider (x, y) the Nash equilibrium of the matrix game M, obtained by e.g. linear programming. Then the Nash method uses x as a probability distribution over random seeds for Black and uses y as a probability distribution over random seeds for White. The algorithm is detailed in Algorithm 2. We also tested a sparse version, which gets rid of pure strategies with low values. The algorithm depends on a parameter α, and it is detailed in Algorithm 3.

C. Criteria

We now give two performance criteria, namely performance against the baseline (which is the original randomized algorithm) and performance against an agent which can choose its seed with perfect rationality among a finite randomly drawn set of a given cardinal - the second criterion is harder, and simulates an opponent who has "learnt" how to play against us by optimizing his seed.

1) Performance against the baseline.: The first criterion is the success rate against the original AI, with its randomized seed. This is precisely the criterion that is optimized in BestSeed; but, obviously, we perform experiments in cross-validation, i.e. the performance obtained by BestSeed and displayed in Section IV is the performance against seeds which were not used in the learning phase.

Algorithm 2 The Nash method for boosting a randomized AI. There is a parameter K; K greater leads to better performance but slower computations. The resulting AI is stochastic. It is outperformed by BestSeed in terms of winning rate against the original (randomized) algorithm, but harder to overfit.

```
Require: $K$ and a randomized AI.
1: for $i \in \{1, \ldots, K\}$ do
2:   for $j \in \{1, \ldots, K\}$ do
3:     Play a game between
         • an AI playing with seed $i$ as Black;
         • an AI playing with seed $j$ as White.
4:     $M_{i,j} \leftarrow 1$ if Black wins, 0 otherwise
5:   end for
6: end for
7: Let $(x, y)$ be a pair of probability distributions over $\{1, \ldots, K\}$, forming a Nash equilibrium of $M$.
8: return The (stochastic) AI using seed
     • $i_0$ randomly drawn with probability distribution $x$ when playing Black
     • and $j_0$ randomly drawn with probability distribution $y$ when playing White.
```

Algorithm 3 The SparseNash method for boosting a randomized AI. Compared to Algorithm 2, there is an additional parameter α.

```
Require: $K$ $\alpha$ and a randomized AI.
1: for $i \in \{1, \ldots, K\}$ do
2:   for $j \in \{1, \ldots, K\}$ do
3:     Play a game between
         • an AI playing with seed $i$ as Black;
         • an AI playing with seed $j$ as White.
4:     $M_{i,j} \leftarrow 1$ if Black wins, 0 otherwise
5:   end for
6: end for
7: Let $(x, y)$ be a pair of probability distributions over $\{1, \ldots, K\}$, forming a Nash equilibrium of $M$.
8: $x_{max} \leftarrow \max_{1 \leq i \leq K} x_i$
9: $y_{max} \leftarrow \max_{1 \leq i \leq K} y_i$
10: For all $i \in \{1, \ldots, K\}$, if $x_i < \alpha x_{max}$, then $x_i \leftarrow 0$.
11: For all $i \in \{1, \ldots, K\}$, if $y_i < \alpha y_{max}$, then $y_i \leftarrow 0$.
12: $x \leftarrow x / \sum_{i=1}^{K} x_i$
13: $y \leftarrow y / \sum_{i=1}^{K} y_i$
14: return The (stochastic) AI using seed
      • $i_0$ randomly drawn with probability distribution $x$ when playing Black
      • and $j_0$ randomly drawn with probability distribution $y$ when playing White.
```

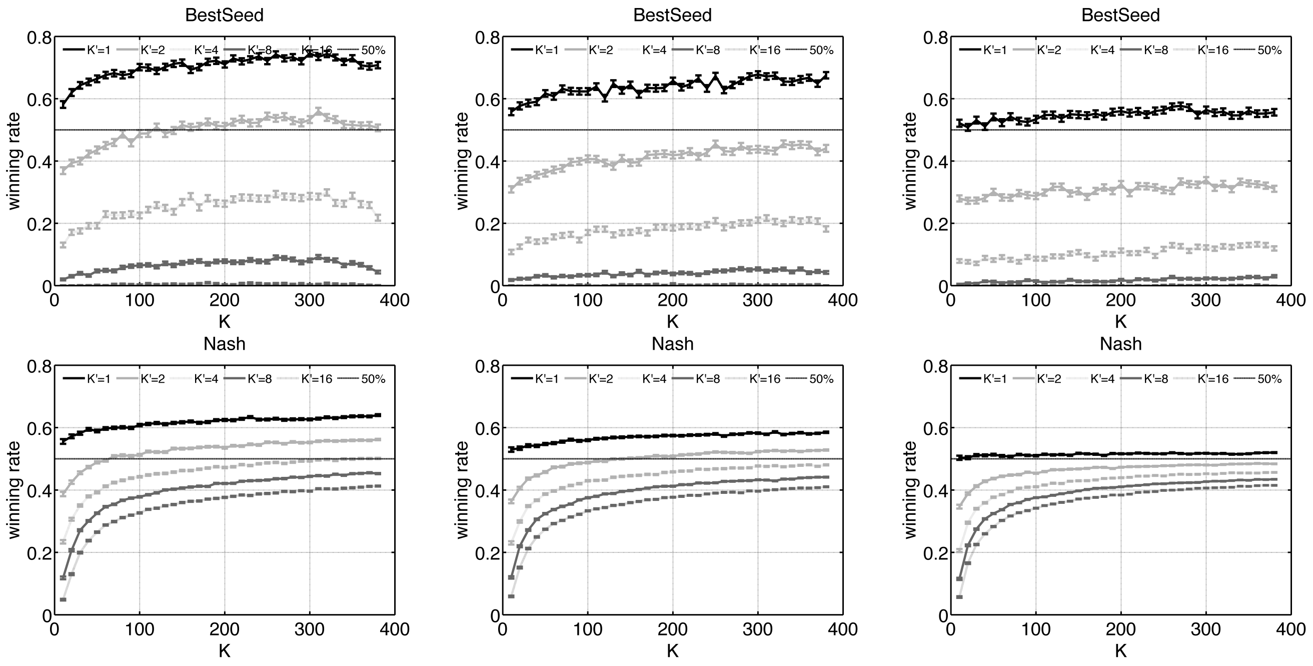

Fig. 1: Winning rate for Phantom Go 5x5 (left), 7x7 (middle) and 9x9 (right). Top: BestSeed; bottom: Nash. X-axis: K such that we learn on a $K \times K$ matrix M. Y-axis: winning rate. The training is performed on a $K \times K$ matrix. The testing is performed on $K' \times K'$ strategies, i.e. K' strategies for Black and K' strategies for White. $K' = 1$ corresponds to a randomized seed - this is therefore the original randomized AI, and performance greater than 50% for $K' = 1$ means that we outperformed the original randomized AI. $K' > 1$ corresponds to the best performing AI, for each color, among K' randomly drawn seeds; this is a very difficult opponent, who can try K'^2 strategies and keep only the best of their results. The black dashed curve refers to a winning rate= 50%. The experiments are repeated 1000 times. The standard deviations are shown in the plots.

2) Performance against an opponent who learns.: The second criterion is the success rate against an opponent who plays with the original randomized AI, but can test K' randomly drawn random seeds and can select the best of these seeds. This modelizes the case in which the opponent can choose (perfect choice) one policy among K' policies. We here consider K' policies, each of them obtained by fixing the random seed to some random value. We do this choice among K' policies for Black, and K' policies for White as well, so that we have indeed the worst performance against $K' \times K'$ policies. This becomes a very tough criterion when K' increases - our opponent can basically test $(K')^2$ openings and choose the one for which we are the weakest.

Obviously, all experiments in the present paper are performed with separate seeds for the learning and the validation experiments, so that no overfitting can explain the positive results in Section IV.

IV. Experimental results

We perform experiments on the Phantom Go testbed. Our randomized AI is Golois [11], [12]. In all our results, we use cross-validation; we test performance against seeds which were not used during the learning phase. We consider values of $K \leq 400$. We use the two criteria described in Section III-C, i.e. winning rate against the original randomized algorithm and worst of the winning rates against $K' \times K'$ deterministic policies obtained as described in Section III-C2. All presented winning rates are the average between the winning rate as Black and the winning rate as White.

We observe in Figure 1 (also Table I):

- The BestSeed approach clearly outperforms the original method in 5x5, 7x7 and 9x9. The performance is excellent in 5x5, greater than 71%; around 67% in 7x7; it is still good in 9x9 (54%).
- The Nash approach reaches 64% in 5x5, 58% in 7x7. This is already reasonably good for a very randomized game such as Phantom Go; in partially observable games like Phantom Go, Poker, or many card games, several games are usually required for knowing the best among two players. In 9x9, we got only 52% - not very impressive.
- The SparseNash approach outperforms BestSeed in terms of success rate against the original randomized AI, in 5x5 (Figure 2 (top), summarized in Table I). Results are however disappointing on larger board sizes (Figure 2 (middle and bottom); also presented in Table I).

These two methods were tested directly on the original algorithm, without using the symmetries of the game or any prior knowledge. All results are obtained with proper cross-validation. Standard deviations are shown on figures and are negligible compared to deviations from 50%. The approach has a significant offline computational cost; but the online

Board	Method		Winning rate (%)				
			$K' = 1$	$K' = 2$	$K' = 4$	$K' = 8$	$K' = 16$
5x5		Baseline	50	30.5 ± 0.9	12.5 ± 0.7	0.5 ± 0.2	0.0 ± 0.0
		BestSeed	70.7 ± 1.0	49.8 ± 1.1	23.4 ± 0.9	4.8 ± 0.4	0.2 ± 0.1
		Nash	63.8 ± 0.3	56.1 ± 0.2	$\mathbf{50.3 \pm 0.2}$	$\mathbf{45.4 \pm 0.2}$	$\mathbf{41.3 \pm 0.2}$
	Sparse	$\alpha = 0.500$	68.3 ± 0.6	56.4 ± 0.6	43.9 ± 0.5	32.8 ± 0.4	24.4 ± 0.3
		$\alpha = 0.750$	$\mathbf{74.7 \pm 0.8}$	$\mathbf{57.7 \pm 0.9}$	36.7 ± 0.9	20.2 ± 0.6	9.0 ± 0.4
		$\alpha = 1.000$	76.2 ± 0.9	55.2 ± 1.1	31.9 ± 1.0	8.7 ± 0.6	0.9 ± 0.2
7x7		Baseline	50	23.0 ± 0.9	8.5 ± 0.6	0.5 ± 0.2	0.5 ± 0.2
		BestSeed	$\mathbf{66.5 \pm 1.0}$	44.1 ± 1.1	19.9 ± 0.9	3.9 ± 0.4	0.1 ± 0.1
		Nash	58.3 ± 0.2	$\mathbf{52.8 \pm 0.2}$	$\mathbf{48.1 \pm 0.2}$	$\mathbf{44.2 \pm 0.1}$	$\mathbf{41.1 \pm 0.1}$
	Sparse	$\alpha = 0.500$	59.6 ± 0.3	51.1 ± 0.3	44.0 ± 0.2	38.7 ± 0.2	33.2 ± 0.2
		$\alpha = 0.750$	58.7 ± 0.6	44.1 ± 0.6	30.9 ± 0.5	21.4 ± 0.4	14.5 ± 0.3
		$\alpha = 1.000$	56.4 ± 1.1	33.0 ± 1.0	13.7 ± 0.8	1.7 ± 0.3	0.0 ± 0.0
9x9		Baseline	50	27.0 ± 1.0	4.0 ± 0.4	1.0 ± 0.2	0.0 ± 0.0
		BestSeed	$\mathbf{54.4 \pm 1.1}$	32.8 ± 1.0	12.2 ± 0.7	2.8 ± 0.4	0.1 ± 0.0
		Nash	51.9 ± 0.1	$\mathbf{48.4 \pm 0.1}$	$\mathbf{45.6 \pm 0.1}$	$\mathbf{43.5 \pm 0.1}$	$\mathbf{41.6 \pm 0.1}$
	Sparse	$\alpha = 0.500$	52.2 ± 0.3	45.3 ± 0.2	39.4 ± 0.2	35.3 ± 0.2	31.1 ± 0.2
		$\alpha = 0.750$	52.4 ± 0.6	38.6 ± 0.5	27.6 ± 0.4	18.4 ± 0.4	12.5 ± 0.3
		$\alpha = 1.000$	52.9 ± 1.1	27.3 ± 1.0	8.2 ± 0.6	1.2 ± 0.2	0.1 ± 0.1

TABLE I: Winning rate for Phantom Go 5x5, 7x7 and 9x9 with $K = 380$ (cf. Figure 1). α is the sparsity parameter (cf. Algorithm 3). The experiments are repeated 1000 times. The standard deviations are shown after $\pm$. $K' = 1$ corresponds to the original algorithm with randomized seed; $K' = 2$ corresponds to the original algorithm but choosing optimally (after checking their performance against its opponent) between 2 possible seeds, i.e. it is guessing, in an omniscient manner, between 2 seeds, each time an opponent is provided. $K' = 4$, $K' = 8$, $K' = 16$ are similar with 4, 8, 16 seeds respectively; $K' = 16$ is a very strong opponent for our original algorithm (our winning rate is initially close to 0), but after Nash seed learning we get results above 40% in 5x5, 7x7 and 9x9.

computational overhead is zero. The offline computational overhead is K^2 times the cost of one game, plus the Nash solving. The Nash solving by linear programming is negligible in our experiments. For large scale matrices, methods such as [17] should provide much faster results as the number of games would be $O(K \log(K)/\epsilon^2)$ instead of K^2 for a fixed precision ϵ.

V. Conclusions

We tested various methods for enhancing randomized AIs by optimizing the probability distribution on random seeds. Some of our methods are not new, but up to now, they were only tested on a fully observable game, without opening book, whereas in fully observable games building an opening book is far less a challenge. We work on Phantom Go, a very challenging problem, with the program which won most competitions in recent years. The three tested methods provide results as follows:

- With BestSeed, we get 71%, 67%, 54% of success rate against the baseline in 5x5, 7x7 and 9x9, just by "managing" the seeds.
- The Nash approach provides interesting results as well, in particular strongly boosting the performance against stronger opponent such as $K' = 2$, $K' = 4$, $K' = 8$, $K' = 16$, reaching 40% (in 5x5, 7x7 and 9x9) whereas our original algorithm was close to 0% winning rate for $K' = 16$. This means that the opening book we have learnt is robust against stronger opponents than the ones used for the self-play involved in our learning.
- Using the Nash approach with sparsity, with the exponent $\alpha = .75$ recommended in earlier papers on sparsity, maybe not the best for each case separately, but outperforming the baseline in all cases.

The method has no computational overhead online - all the computational cost is an offline learning. As a consequence, the method looks like a free bonus: when your randomized AI is ready, apply Algorithm 2 and get a better AI. The BestSeed method is the best performing one, but it can be overfitted. The Nash approach is less efficient against the original AI, but more robust, i.e. more difficult to overfit.

Further work

We propose the following further works:

- The approach is quite generic, and could be tested on many games in which randomized AIs are available. For the BestSeed approach, the game does not have to be a two-player game.
- Our work does not use any of the natural symmetries of the game; this should be a very simple solution for greatly improving the results; in particular, it would be much harder to overfit BestSeed if it was randomized by the 8 classical board symmetries.
- Mathematically analyzing the approach is difficult, because we have no assumption on the probability distribution of $\mathbb{E}_j M_{i,j}$ for a randomly drawn seed i - how many i should we test before we have a good probability of having a really good one ? Bernstein inequalities [19], [20], [21] for the BestSeed approach, and classical properties of Nash equilibria for the Nash approach, provide only preliminary elements.
- Computing approximate Nash equilibria (using [17] or [18]) should strongly reduce the offline computational cost. The computational cost was not a big deal for the results presented in the present paper, but performance might be much better with K larger. Approximate Nash equilibria do not need the entire $K \times K$ matrix; they

only sample $O(K \log(K)/\epsilon^2)$ elements of the matrix for a precision ϵ.

- This last further work opens some problems also in the algorithmic theory of Nash equilibria. We have done the present work in a not anytime manner; we know K a priori, and we do not have any approximate results as long as the K^2 games are not played. However, we might prefer not to choose a priori a number K of games, and get anytime approximate results. To the best of our knowledge, [17], [18] have never been adapted to an infinite set of arms. Also, adversarial bandit approaches such as Exp3 [18] have never been parallelized. [17] is parallel, but possibly harder to adapt in an anytime setting.

References

[1] E. V. Nalimov, C. Wirth, G. M. Haworth *et al.*, "Kqqkqq and the kasparov-world game," *ICGA Journal*, vol. 22, no. 4, pp. 195–212, 1999.

[2] R. Gaudel, J.-B. Hoock, J. Pérez, N. Sokolovska, and O. Teytaud, "A principled method for exploiting opening books," in *Computers and Games*. Springer, 2010, pp. 136–144.

[3] G. Chaslot, M. Winands, I.Szita, and H. van den Herik, "Parameter tuning by cross entropy method," in *European Workshop on Reinforcement Learning*, 2008. [Online]. Available: http://www.cs.unimaas.nl/g.chaslot/papers/ewrl.pdf

[4] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, pp. 72–83, 2006.

[5] L. Kocsis and C. Szepesvari, "Bandit based Monte-Carlo planning," in *15th European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.

[6] B. Bruegmann, "Monte-carlo Go (unpublished draft http://www.althofer.de/bruegmann-montecarlogo.pdf)," 1993.

[7] B. Bouzy, "Associating domain-dependent knowledge and monte carlo approaches within a go program," *Information Sciences, Heuristic Search and Computer Game Playing IV, Edited by K. Chen*, no. 4, pp. 247–257, 2005.

[8] B. Bouzy and G. Chaslot, "Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go," *In G. Kendall and Simon Lucas, editors, IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK*, pp. 176–181, 2005.

[9] T. Cazenave, "Nested monte-carlo search," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 456–461.

[10] J. Méhat and T. Cazenave, "Combining uct and nested monte carlo search for single-player general game playing," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 2, no. 4, pp. 271–277, 2010.

[11] T. Cazenave, "A phantom-go program," in *Proceedings of Advances in Computer Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers, Eds., vol. 4250. Springer, 2006, pp. 120–125.

[12] T. Cazenave and J. Borsboom, "Golois wins phantom go tournament," *ICGA Journal*, vol. 30, no. 3, pp. 165–166, 2007.

[13] D. L. Saint-Pierre and O. Teytaud, "Nash and the Bandit Approach for Adversarial Portfolios," in *CIG 2014 - Computational Intelligence in Games*, ser. Computational Intelligence in Games, IEEE. Dortmund, Germany: IEEE, Aug. 2014, pp. 1–7. [Online]. Available: https://hal.inria.fr/hal-01077628

[14] J. V. Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton University Press, 1944. [Online]. Available: http://jmvidal.cse.sc.edu/library/neumann44a.pdf

[15] J. Nash, "Some games and machines for playing them," Rand Corporation, Tech. Rep. D-1164, 1952.

[16] B. von Stengel, "Computing equilibria for two-person games," in *Handbook of Game Theory*, R. Aumann and S. Hart, Eds. Amsterdam: Elsevier, 2002, vol. 3, pp. 1723 – 1759.

[17] M. D. Grigoriadis and L. G. Khachiyan, "A sublinear-time randomized approximation algorithm for matrix games," *Operations Research Letters*, vol. 18, no. 2, pp. 53–58, Sep 1995.

[18] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: the adversarial multi-armed bandit problem," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 322–331.

[19] S. Bernstein, *The Theory of Probabilities*. Gastehizdat Publishing House, Moscow, 1946.

[20] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, pp. 13–30, 1963.

[21] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Math. Stat.*, vol. 23, pp. 493–509, 1952.

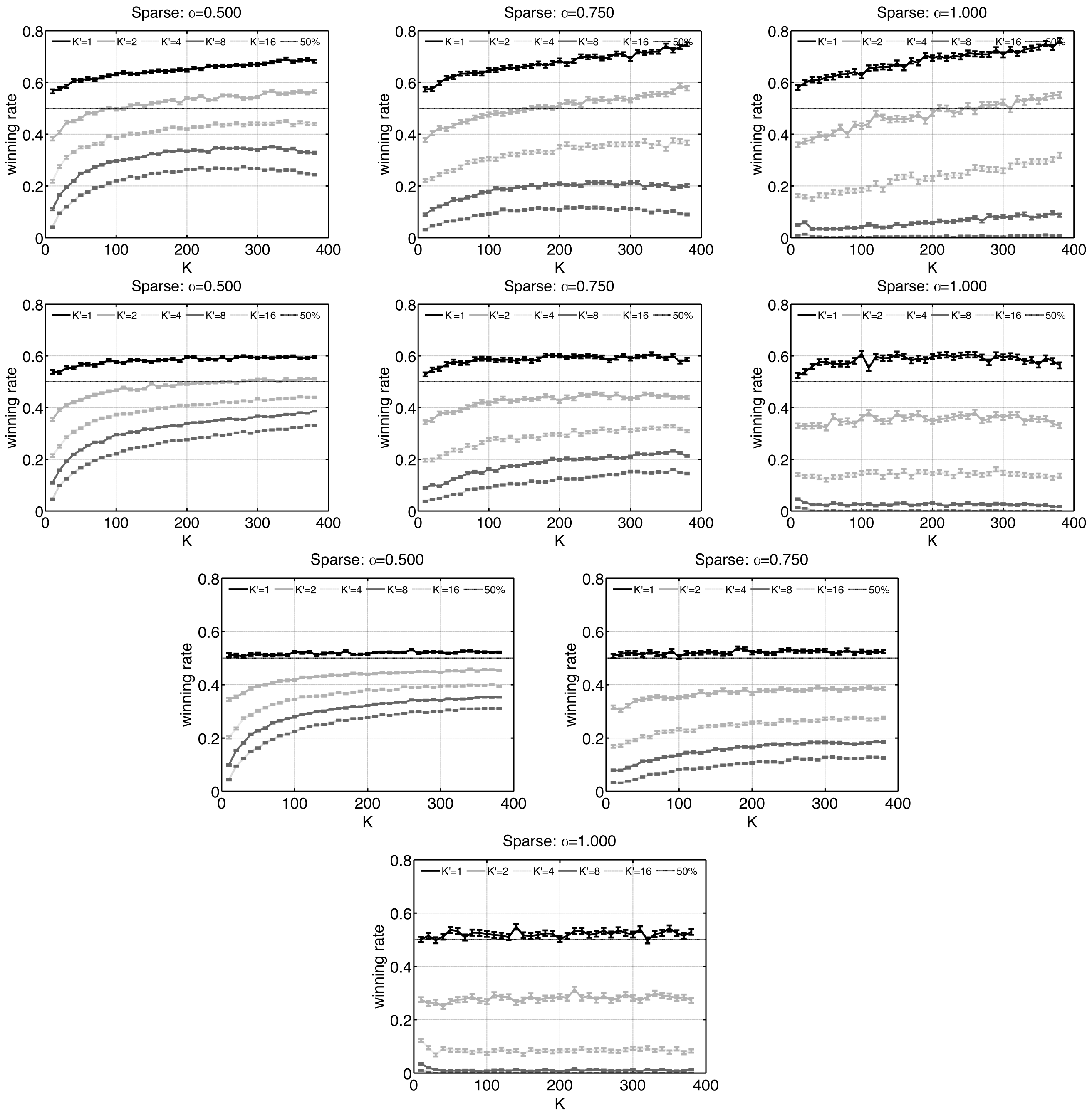

Fig. 2: Winning rate for Phantom Go 5x5 (top), 7x7 (middle) and 9x9 (bottom) using sparse strategy with different sparsity parameter α. X-axis: parameter K (size of the learning set). Y-axis: performance in generalization against the original algorithm ($K' = 1$) and against the learning opponent (see Section III-C2; $K' = 2$ to $K' = 16$).