Nash and the Bandit Approach for Adversarial Portfolios

David L. Saint-Pierre, Olivier Teytaud

Nash and the Bandit Approach for Adversarial Portfolios

David L. St-Pierre
TAO (Inria), LRI,
Univ. Paris-Sud
Paris, France
Email: davidls@lri.fr

Olivier Teytaud
TAO (Inria), LRI,
Univ. Paris-Sud
Paris, France
Email: olivier.teytaud@inria.fr

***Abstract*—In this paper we study the use of a portfolio of policies for adversarial problems. We use two different portfolios of policies and apply it to the game of Go. The first portfolio is composed of different versions of the GnuGo agent. The second portfolio is composed of fixed random seeds. First we demonstrate that learning an offline combination of these policies using the notion of Nash Equilibrium generates a stronger opponent. Second, we show that we can learn online such distributions through a bandit approach. The advantages of our approach are (i) diversity (the Nash-Portfolio is more variable than its components) (ii) adaptivity (the Bandit-Portfolio adapts to the opponent) (iii) simplicity (no computational overhead) (iv) increased performance. Due to the importance of games on mobile devices, designing artificial intelligences for small computational power is crucial; our approach is particularly suited for mobile device since it create a stronger opponent simply by biasing the distribution over the policies and moreover it generalizes quite well.**

I. Introduction

The use of portfolio is at the center of a revolution in machine learning [1], [2], Artificial Intelligence, planning, combinatorial optimization [3], [4], [5] and games [6], [7], [8]. "Portfolio" here refers to the family of algorithms used in the solving, whereas "portfolio combination" or "combination" refers to the combined algorithm. There exist several ways to build such combination. It can be combined either "externally", which refers to a process that do not explicitly enters in each algorithm, through "chaining" [9], which means interrupting one and using its state for another algorithm or even "internally" [10]. The most famous applications of portfolios are around SAT-solving[11].

In this paper, we focus on portfolio of policies. Portfolios of policies have been less widely explored, except for e.g. combinations of local controllers by Fuzzy Systems[12], Voronoi controllers[13] or some case-base reasoning[14]. These methods are based on "internal" combinations, using the current state for choosing between several policies. We will here focus on external combinations. Such combinations are sometimes termed "ensemble methods"; however, we simply consider weighted averages of already constructed policies, the simplest case of ensemble methods.

A distribution over a portfolio can be computed either offline [15] or online [16], [17]. In this paper, we use two different methods. First we compute a Nash Equilibrium (NE) over the portfolio of policies in an offline fashion. Second, we compute online a distribution using a bandit approach. The second case is adaptive, the coefficients will depend on the opponent's decisions.

The main contribution of this paper is to propose a methodology that can generically improve the performance of policies without actually changing the policies themselves, except through the policy's options or even the policy's random seed. Incidentally, we establish that the random seed can have a contribution, just because it has an effect on rote learning; while a fixed random seed cannot be strong against an adaptive opponent, our policies are more diversified (for the offline portfolio that we propose, which combines several seeds) or adaptive (for the online portfolio that we propose). It is particularly well suited when the computational power is limited. We study 2 different portfolios of policies:

- The first one is a set of algorithms that can play the game of Go (actually, different options of a same program). The goal here is to find a probability distribution for choosing among these algorithms.
- The second portfolio is composed of a set of random seeds for a given algorithm that plays the game of Go. The difficulty here lies in finding a distribution over the set of random seeds such that it provides a strong strategy either versus a known opponent (an opponent for which we know the set of random seeds) or an unknown one.

In all cases we show that there is a way to compute a distribution over the portfolio such that it generates a robust (not exploitable) agent: this is our Nash-portfolio. Moreover we show that we can learn a specialized distribution, adaptively, given a fixed stationary opponent. This is our UCBT-portfolio.

The rest of the paper is divided as follows. Section II formalizes the settings. Section III describes our approach. Section IV presents the results and Section V concludes.

II. Problem Statement

A policy π is defined as follows. Let us consider a state space S, an initial state $s_0 \in S$ and a transition function $(s, a) \mapsto (s', r)$ where r is the reward when playing action a in state s. A policy $\pi(\cdot)$ selects an action $a \in A_s$ given the current state $s \in S$, where A_s is the set of legal actions given the state s. The resulting state being s'. In several cases,

the reward may not be directly observable. By repeating this process until a final state $s_f \in S_f \subseteq S$ is reached, where S_f is the set of final states, then the reward can be retrieved.

Moreover, we are interested in adversarial portfolios. This means that more than one player can intervene in the outcome. We limit ourselves to constant-sum 2-player games. In the 2-player case, each state s is equipped with a player index, stating whether, in state s, it is player 1's turn to play or player 2's turn to play. A simulation starts at s_0, then decisions are made by players, until a final state s_f is reached. Thus, to get a reward r, we must know the initial state s_0, the policy π_1 for player 1 and the policy π_2 for player 2.

We consider here matrix games with sum equal to 1 instead of an arbitrary constant, without loss of generality. Consider a matrix $K \times K'$, with values in $[0, 1]$. This matrix models a game as follows:

- Simultaneously and privately:
  - Player 1 chooses $i \in \{1, \dots, K\}$.
  - Player 2 chooses $j \in \{1, \dots, K'\}$.
- Then they receive rewards as follows:
  - Player 1 receives reward $M_{i,j}$.
  - Player 2 receives reward $1 - M_{i,j}$.

A pure strategy (for player 1) consists in playing a given, fixed $i \in \{1, \dots, K\}$, with probability 1. A mixed strategy, or simply a strategy, consists in playing i with probability p_i, where $\sum_{i=1}^{K} p_i = 1$ and $\forall i \in \{1, \dots, K\}, p_i \geq 0$. Pure and mixed strategies for player 2 are defined similarly. Pure strategies are a special case of mixed strategies.

It is known since [18] that there exist strategies p and q for the first and second player respectively, such that

$$\forall (p', q'), p'Mq \leq pMq \leq pMq'. \tag{1}$$

p and q are not necessarily unique, but the value $v = pMq$ is unique and it is, by definition, the value of the game. The exploitability of a strategy p for the first player is $exploit_1(p) = v - \min_q pMq$. $exploit_1(p) = 0$ is equivalent to the fact that p is the first-player part of a Nash equilibrium (p, q). The exploitability of a strategy q for the second player is $exploit_2(q) = \max_p pMq - v$ and it verifies similar properties.

III. Approach

Section III-A explains how to combine policies offline, given a set of policies for player 1 and a set of policies for player 2. Section III-B explains how to combine policies online, given a portfolio of policies for player 1 and given an opponent.

A. Nash-portfolios: combining policies

Consider two players P1 and P2, playing some game (not necessarily a matrix game). Assume that P1 has a portfolio of K policies. Assume that P2 has a portfolio of K' policies. Then, we can construct a static combination of these policies by solving (i.e. finding a Nash equilibrium of) the matrix game associated to the matrix M, with $M_{i,j}$ the winning rate of the i^{th} policy of P1 against the j^{th} policy of P2. Solving this 1-sum matrix game provides $p_1, \dots, p_K$ and $q_1, \dots, q_{K'}$, probabilities, and the combination consists in playing, for P1, the i^{th} policy with probability p_i and, for P2, the j^{th} policy with probability q_j. Such a combination will be termed here a Nash-portfolio. By construction,

- the Nash-portfolio can play both as Black and as White;
- the Nash-portfolio does not change over time but is, in the general case, stochastic.

Definition: *given a set P_1 of K policies for player 1 and a set P_2 of K' policies for player 2,*

- *the strategy (as player 1) playing the i^{th} strategy for player 1 with probability p_i;*
- *and the strategy (as player 2) playing the i^{th} strategy for player 2 with probability q_i;*

is termed the Nash-portfolio of (P_1, P_2) if the p_i and q_i are defined as solutions of Eq. 1.

The Nash equilibrium can be found using an exact solving of the matrix M[19]. It can also be found approximately and iteratively, in sublinear time, as shown by [20], [21]; the EXP3 algorithm is classical for doing so. Coevolution is also a possibility [22], [23], [24].

From the properties of Nash equilibria, we deduce that the Nash-portfolio has the following properties:

- It depends on a family of policies for player 1 *and* on a family of policies for player 2. It is therefore based on a training, as in, e.g., [15] in the framework of optimization.
- It is optimal (for player 1) among all combinations of the pure strategies in the portfolio of player 1, in terms of both
  - worst case among the pure strategies in the portfolio of player 2;
  - worst case among the mixed strategies over the portfolio of player 2.
- It is not necessarily uniquely defined.

In optimization settings, it is well known[25] that having a somehow "orthogonal" portfolio of algorithms, i.e. algorithms as different from each other as possible, is a good solution for making the combination efficient. It is however difficult, in the context of policies, to know in advance if two algorithms are orthogonal.

B. UCBT-Portfolio

Section III-A assumed that P_1 and P_2, two sets of strategies, are available, and that we want to define a combination of policies in P_1 (resp. in P_2). A different point of view consists in adapting online the probabilities p_i and q_i, against a fixed opponent. We propose the following algorithm (in the case of player 1 having K policies at hand), directly inspired by the bandit literature[26], [27], and, more precisely, by Upper-Confidence-Bounds-Tuned (UCBT)[28]:

- Define $n_i = 0$, $r_i = 0$, for $i \in \{1, \dots, K\}$.
- For each iteration $t \in \{1, 2, 3, \dots\}$.
  - compute for each $i \in \{1, \dots, K\}$ $score(i) = \min(1, r_i/n_i + \frac{1}{100}\sqrt{C \log(4^{pt})/n_i}$

$+\frac{16}{100}\log(4^{pt})/n_i)$. using $X/0 = +\infty$ (even for $X = 0$), $p = 2.1$ and $C = 2$ (UCBT, i.e. UCB-Tuned, formula).
- choose k maximizing $score(k)$.
- play a game using algorithm k in the portfolio.
- if it is a win, $r_i \leftarrow r_i + 1$.
- $n_i \leftarrow n_i + 1$.

Definition. *We refer to this adaptive player as UCBT-Portfolio, or Bandit-Portfolio.*

IV. Experiments

The game of Go is an ancient oriental game, born in China probably at least 2500 years ago. It is still a challenge for artificial intelligence, as even though Monte-Carlo Tree Search (MCTS[29]) revolutionized the domain, the best programs are still far from the professional level. Go is a very deep game[30], and professional Go players are still able to win games against the best Go programs in 9x9.

Section IV-A focuses on a portfolio of different algorithms (actually, variants of a same program, using different options) and explores the performance of our approach on this setting. Next, Section IV-B studies our approach but this time over a portfolio of deterministic policies, obtained by fixing random seeds in a randomized program. Thanks to very low computational cost, both settings are relevant for the application of our approach on mobile devices.

A. Portfolio of algorithms

1) Test case: variants of a program: We consider the problem of combining several options, leading to several variants (each variant corresponds to a set of options which are enabled), of an artificial intelligence for the game of Go.

Our matrix M is a 32×32 matrix, where $M_{i,j}$ is the winning rate of the i^{th} variant of GnuGo (as black) against the j^{th} variant of GnuGo (as white). We consider all combination of 5 options of GnuGo, hence $32= 2^5$ variants. In short, the first option is 'cosmic-gnu', which focuses on playing at the center, thus usually weakening the AI. The second option is the use of fuseki (global opening book). The third option is 'mirror', which consist in mirroring your opponent at the early stage of the game. The fourth option is the large scale attack, which evaluates if a large attack across several groups is possible. The fifth option is the break-in. It consists in breaking the game into territories that require deeper tactical reading and are impossible to detect otherwise. It revises the territory valuations. Further details on the 5 options are listed on our website https://www.lri.fr/~teytaud/games.html.

In this section, experiments are performed on the convenient 7x7 framework, with MCTS having 300 simulations per move - this setting is consistent with the mobile device setting.

After extracting the matrix M by intensive experiments on a cluster, we solve the matrix game associated to M and get a portfolio for Black and a portfolio for White.

In this section we refer to the i^{th} algorithm for Black as BAIi (Black Artificial Intelligence # i), and WAIj is the j^{th} algorithm for White.

2) Nash-based portfolio: We here check the exploitability of our method, compared to other combinations. The other considered combinations are:

- Uniform combination: each pure strategy is equally likely.
- Maximal pure combination: choose the pure strategy which maximizes the average reward against the uniform combination.

By definition, the exploitability of the Nash-portfolio method is zero, because it is a Nash equilibrium. But other methods usually have a positive exploitability. We here quantify this phenomenon. We get the following on the data set described above (variants of a program):

- Value of the 32×32 matrix game: 61.18% (i.e. the game is easier for Black).
- In the NE, the number of selected options is 5 for Black and also 5 for White, i.e. roughly 15.6% of the variants are in the Nash.
- Against a uniform random combination of the 32 options, the winning rate of the Nash-portfolio as Black (resp. White) is 65.52% (resp. 49.71%), i.e. the Nash-portfolio outperforms the uniform random combination by 4.35% as Black (resp. 10.9% as White).
- Exploitability of Portfolios with one option only (selected as the best on average):
  - Exploitability of the Portfolio restricted to BAIi, with $i = \arg\max \sum_{j=1}^{K'} M_{i,j}$: 3.15%.
  - Exploitability of the Portfolio restricted to WAIj, with $j = \arg\min \sum_{i=1}^{K} M_{i,j}$: 5.85%.

Conclusion: We conclude that our Nash-combination is moderately better than each combination of options considered separately. This section is devoted to experiments limited to a learning set of 32 combinations of options for each player, and therefore could be considered as an overfit; we will check the efficiency of our Nash-Portfolio in generalization in Section IV-B4, on the second test case (random seeds), for which results of the Nash portfolio are better.

3) UCBT Portfolio: Here we present the losing rate of our UCBT Portfolio against 2 baselines. The first baseline is the Nash equilibrium (label $Nash$ in Figure 1), which consists in computing the Nash portfolio as in the previous section. The second baseline is the uniform player (label $Unif$) which consists in playing each variant with the same probability.

Here, we apply our UCBT Portfolio for learning

- As White against Nash-portfolio.
- As Black against Nash-portfolio.
- As White against each pure Black policy.
- As Black against each pure White policy.

Losing rates of the recommended variant are presented in Fig. 1. All results are averaged over 1000 runs. The X-axis shows the number of iterations of UCBT (i.e. number of played games for learning) whereas the Y-axis represents the frequency at which the game is lost.

Conclusion: We see that (i) UCBT eventually reaches, against Nash-portfolio, approximately the value of the game for each player (ii) the Nash-portfolio is among the most

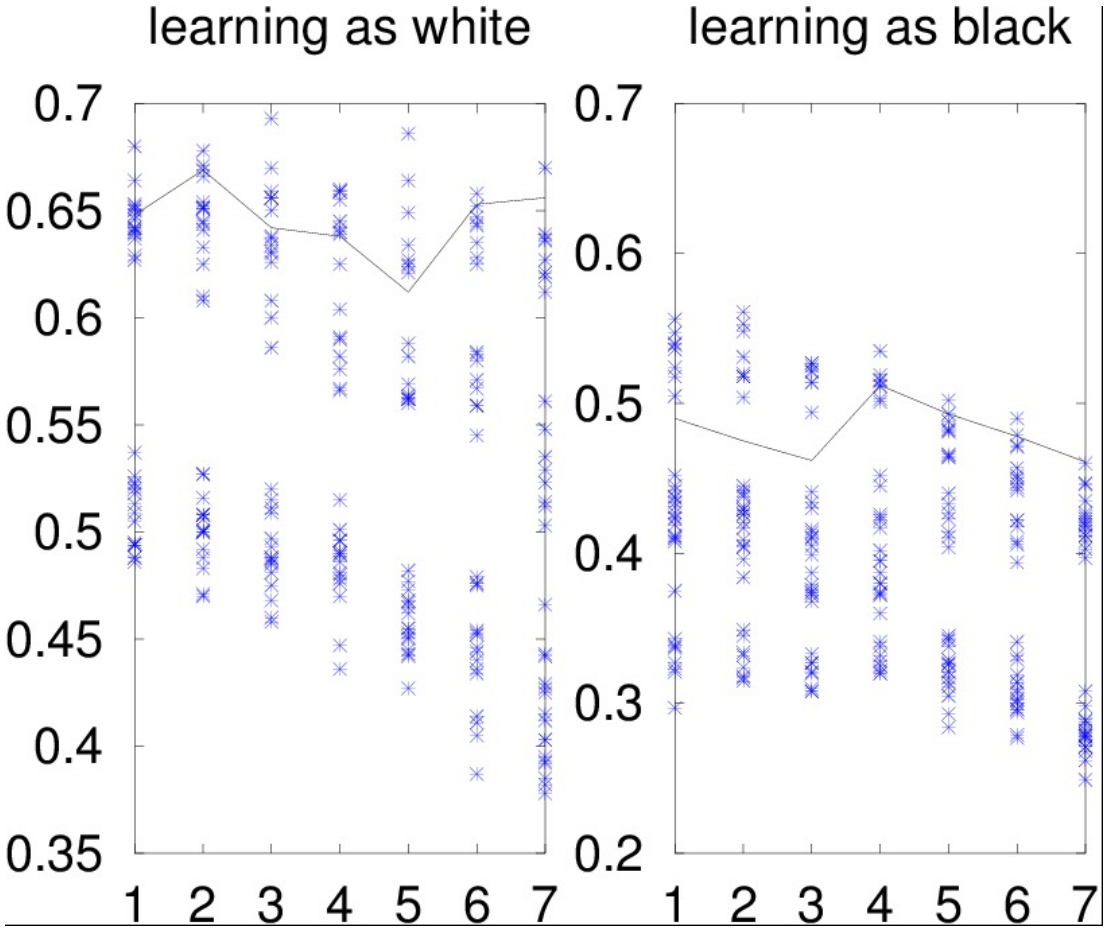


Fig. 1: Losing rate of UCBT-portfolio, versus the online learning time, against (i) Nash-Portfolio (black line) (ii) each variant independently (stars). X-axis: log2(number of iterations of UCBT (i.e. number of played games for learning). Y-axis: frequency at which the game is lost. In this case, the variants under work are GnuGo variants (different options). All experiments are reproduced 1000 times.

difficult opponents (the curve decreases slowly only). (iii) each deterministic combination of options is clearly beaten after some runs, which shows the adaptivity of UCBT Portfolio and therefore its efficiency as a eTeacher.

B. Portfolio of Random Seeds

1) Test case: The previous section combines existing algorithms. Now, this section (i) creates variants thanks to different random seeds and (ii) combine them. This means that $M_{i,j}$ is 1 if, with random seed i, Black wins against White with random seed j. Importantly, the number of games to be played for getting the matrix involved in the Nash-portfolio (Eq. 1) is K^2, with K the number of random seeds under tests - no need for playing multiple games as random seeds make it deterministic. For making the random seed principle more convincing, we switch to 9x9, which is harder than 7x7 in terms of rate-learning.

[6] proposed a combination of opening book, using tools similar to those we propose in Section III-A for combining policies. We here propose to simply use GnuGo's random seed for having several Gnugo variants (hence, programs playing the opening differently). The random seed of GnuGo makes the program deterministic, by fixing the seed used in all random parts of the algorithm. We define 32 variants, using "GnuGo –level 10 –random-seed V" with V in $\{1, \ldots, 32\}$. In other words, we use a MCTS with 80 000 simulations per moves.

2) Nash-based portfolio: We here check the exploitability of our method, compared to other combinations. By definition, the exploitability of the Nash-portfolio method is zero, because it is a Nash equilibrium. But other methods are exploitable. We here quantify this phenomenon. We get the following on the data set described at the beginning of Section IV:

- Value of the game: 54.16% (i.e. the game is easier for Black).
- The winning rate at the Nash equilibrium for the Black player is incidentally the value of the game 54.16%.
- In the NE, the number of selected options is 11 for Black and 9 for White. This indicates that roughly $\frac{1}{3}$ of the random seeds are relevant.
- Against a uniform random combination of the 32 options,

the winning rate of the Nash-portfolio as Black (resp. White) is 68.51% (resp. 38.80%), i.e. the exploitability of the uniform random combination is 14.35% (resp. 15.36%).

- Exploitability of Portfolios with one option only:
  - Exploitability of the Portfolio restricted to BAIi, with $i = \arg\max \sum_{j=1}^{K'} M_{i,j}$: 44.24%.
  - Exploitability of the Portfolio restricted to WAIj, with $j = \arg\max \sum_{i=1}^{K} M_{i,j}$: 33.38%.

Conclusion: Nash-portfolios seems useful even in cases in which an optimal deterministic policy exists, such as fully observable games (Go is such a case). The generalization ability will be tested in Section IV-B4.

3) Learning UCBT: Here we present the losing rate of UCBT against 3 baselines. The first baseline is the Nash equilibrium (label $Nash$ and previously defined in Section III-A. The second baseline is the uniform player (label $Unif$) which consists in playing uniformly each option of the bandit. The third baseline consists in playing a single deterministic strategy (only one random seed) regardless of the opponent.

Figure 2 presents the results. The x-axis represents the number of iterations (on logarithmic scale) of UCBT (i.e. number of played games for learning). The y-axis is the frequency at which the game is lost. All results are averaged over 1000 runs.

First and foremost, as the number of iterations grows, there is a clear learning against both $Nash$ and $Unif$ baselines. We see that (i) UCBT eventually reaches, against Nash-portfolio, approximately the value of the game for each player (ii) the Nash-portfolio is among the most difficult opponents (the curve decreases slowly only). We can also observe from Figure 1 that against the $Unif$ baseline UCBT learns a strategy that outperforms this opponent.

When it plays as the Black player, it takes less than 2^7 (128) games to earns the correct strategy and wins with a 100 % ratio against every single deterministic variant. As the White player, it is even faster with only 2^5 games required to always win.

Conclusion: UCBT can learn very efficiently against a fixed deterministic opponent; this confirms its ability as a eTeacher. It performs better than Nash-portfolio against Uniform, showing that even against a stochastic opponent it can perform well, and in particular better than the Nash. This is not a contradiction with the Nash optimality; the Nash portfolio is optimal in an agnostic sense, whereas UCBT tries to overfit its opponent and can therefore exploit it better.

4) Generalization ability of Nash-Portfolio: Here we are interested in the generalization of a Nash-portfolio. In other words, whether it is possible to use a distribution computed over a portfolio of policies (learnt against a given set of opponent policies) against new opponent policies that were not part of the initial matrix. The idea is to select a submatrix of size N, choose a combination in this submatrix (i.e. this is the learning set) and make it play against the remainder of the seeds (i.e. this is the validation set).

Figure 3 presents the results of 3 different approaches. The first approach, labeled $Nash$, is to use the distribution representing the NE computed on the initial submatrix against the new opponent policies. The second approach, labeled $BestArm$ is to select the single best performing policies. The third, our baseline, is to use the uniform distribution $Unif$ over the initial set of seeds. The x-axis represents the number of policies N considered (hence a matrix $M = N^2$). The y-axis shows the win rate of the different approaches. All experiments are reproduced 1 000 times.

From Figure 3 we can observe that there is a clear advantage to use either the $Nash$ or the $BestArm$ approach when facing a new set of policies. Moreover, as expected, as the size of the initial matrix grows, the winning rates of both $Nash$ and $BestArm$ increase when compared to the baseline.

It does not come as a surprise that the approach $BestArm$ performs slightly better than the $Nash$ against a uniformly random opponent. The $BestArm$ approach is particularly well suited to play against such an opponent. However, the $BestArm$ policy is easily exploitable.

Figure 4 shows the difference between a $Nash$ policy and a $BestArm$ policy in terms of exploitability. The x-axis represents the number of policies considered. The y-axis shows the loss rates. All experiments are reproduced 100 times.

From Figure 4 it clearly appears that $BestArm$ is a strategy very easy to exploit. Thus, even if Figure 3 shows that the use of the $BestArm$ policy outperforms $Nash$ versus the uniform baseline, $Nash$ is a much more resilient strategy.

Conclusion: The Nash portfolio and the $BestArm$ portfolio provides a stable solution which generalizes against new opponents. The Nash portfolio is by definition optimal as a worst case against a given set of opponents, but the $BestArm$ performs better in generalization against a fixed uniform portfolio.

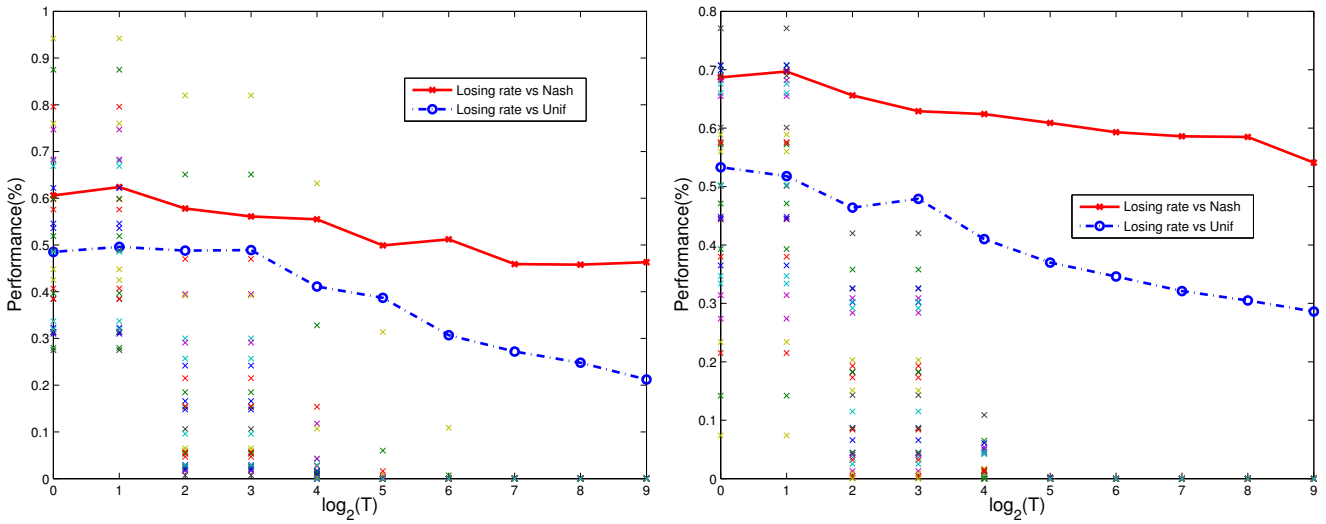
V. Conclusion

We proposed two algorithms for combining policies:

- The Nash-Portfolio, which learns offline, given a family of variants for each player;
- The Bandit(UCBT)-Portfolio, which learns online, given an opponent.

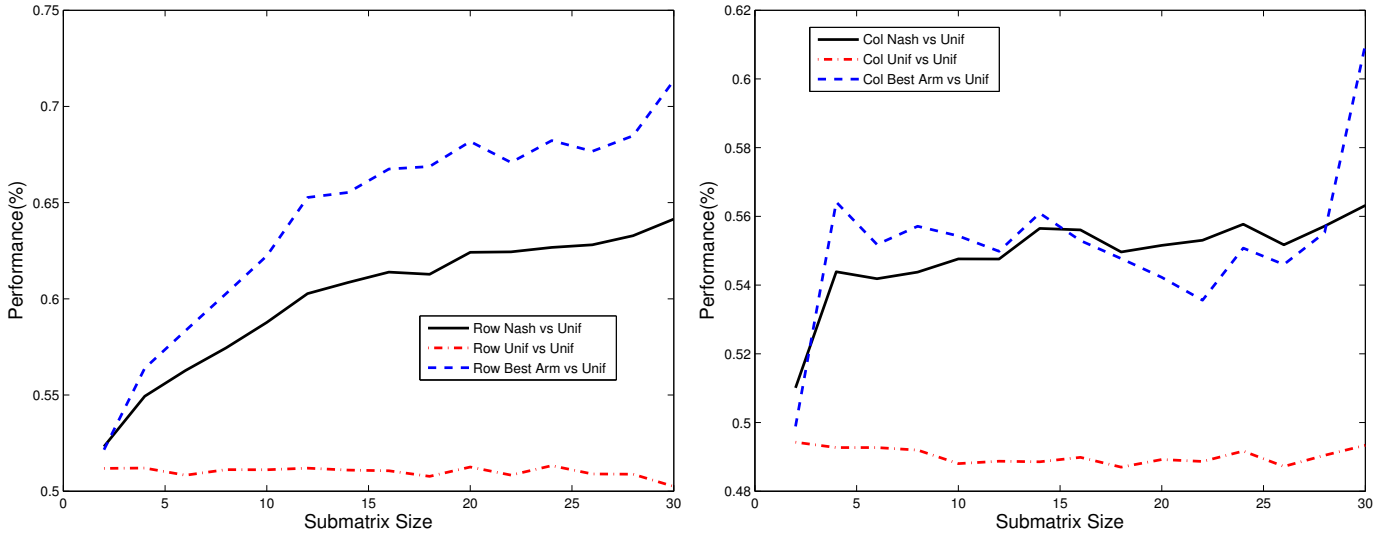
We have seen that:

- The Nash-Portfolio is more diversified than any of its components; it is harder to learn against than any of its components and harder to learn against than the uniform-Portfolio;
- The UCBT-Portfolio can learn a combination until reaching the exploitability of a stationary opponent (this is mathematically guaranteed by properties of UCBT, which is a consistent bandit algorithm in the discrete setting). In particular, it defeated clearly the deterministic variants in Fig. 2, reaching 100% winning rate. It also performs quite well against the default policy, which is uniformly randomized random seed. Only the Nash-Portfolio resists much better, which shows that we can enhance a randomized algorithm a lot, just by biasing the choice of the random seed.



(a) Black (b) White

Fig. 2: Losing rate of UCBT-portfolio, versus the online learning time, against (i) Nash-Portfolio (black line) (ii) each option independently (stars). X-axis: log2(number of iterations of UCBT (i.e. number of played games for learning). Y-axis: frequency at which the game is lost. In this case, the variants under work are random seeds. All experiments are reproduced 1000 times.



(a) Black (b) White

Fig. 3: Winning rate of 3 baselines against the uniform strategy in generalisation. The x-axis represents the number of policies considered. The y-axis shows the win rates. All experiments are reproduced 1000 times.

The Nash-portfolio generalizes efficiently to new opponents, as shown by Section IV-B4. Therefore our tools provide an easy improvement, on top of randomized algorithms equipped with a random seed, or on top of variants of an algorithm.

The computational cost could potentially become an issue in Section IV-A (as we averaged multiple games for building the matrix). It is not the case in Section IV-B where deterministic games are used. It should also be pointed out that solving Nash was fast, and if needs be, we can further the speed of the computation using algorithms that can ϵ-approximate a NE in sublinear time[20], [21] - so that a number of games linear in $\max(K, K')\log(KK')/\epsilon^2$ (less than the number $K \times K'$ of elements in the matrix) is sufficient.

Results are therefore both

- An improvement in terms of eTeaching; our UCBT-Portfolio algorithm is more difficult to overfit, more diversified, than the uniform random seed, and is adaptive. This does not involve any additional computing power as UCBT is online and has a negligible internal cost.
- An improvement in terms of adaptivity; our Nash-

Portfolio is harder to overfit. At the end of the offline computation of the Nash equilibrium, it is just a bias in the random seed distribution, so the additional computational cost is negligible. The non-intuitive key point in the "random seed" part of this work is that biasing the random seed has an impact.

- An improvement in terms of playing strength, as our Nash-Portfolio performs better than each of its components.

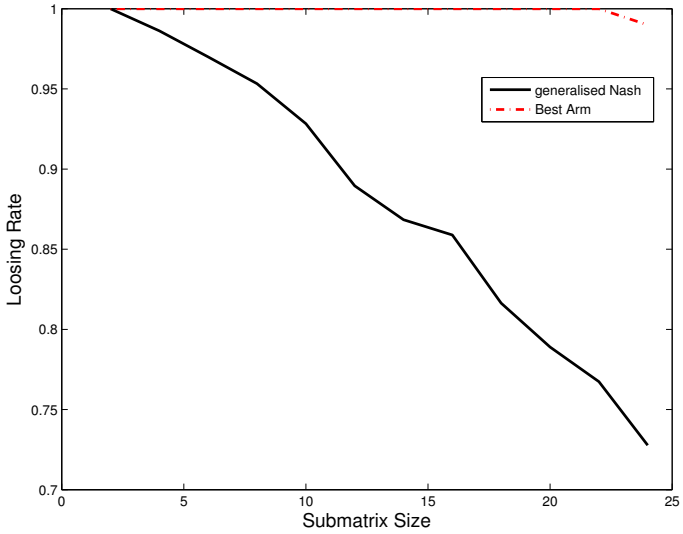


Fig. 4: Loosing rate of the $Nash$ and $BestArm$ policies against an opponent that knows their respective strategy. The x-axis represents the number of policies considered. The y-axis shows the loss rates. All experiments are reproduced 100 times.

REFERENCES

[1] P. E. Utgoff, "Perceptron trees: A case study in hybrid concept representations," in *National Conference on Artificial Intelligence*, 1988, pp. 601–606.

[2] D. W. Aha, "Generalizing from case studies: A case study," in *Proceedings of the 9th International Workshop on Machine Learning*. Morgan Kaufmann Publishers Inc., 1992, pp. 1–10.

[3] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding random sat: beyond the clauses-to-variables ratio," in *Principles and Practice of Constraint Programming CP 2004*, M. Wallace, Ed., vol. 3258 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 438–452.

[4] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Hydra-mip: automated algorithm configuration and selection for mixed integer programming," in *RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[5] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *CoRR*, vol. abs/1210.7959, 2012.

[6] R. Gaudel, J.-B. Hoock, J. Pérez, N. Sokolovska, and O. Teytaud, "A principled method for exploiting opening books," in *Computers and Games*. Springer, 2011, pp. 136–144.

[7] B. Bouzy, M. Métivier, and D. Pellier, "Hedging algorithms and repeated matrix games," in *ECML Workshop on Machine Learning and Data Mining In and Around Games*, 2011.

[8] M. Swiechowski and J. Mandziuk, "Self-adaptation of playing strategies in general game playing," *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, 2014.

[9] J. Borrett, E. P. K. Tsang, and C. C. Sq, "Towards a formal framework for comparing constraint satisfaction problem formulations," 1996.

[10] V. Vassilevska, R. Williams, and S. L. M. Woo, "Confronting hardness using a hybrid approach," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. ACM, 2006, pp. 1–10.

[11] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: Portfolio-based algorithm selection for sat." *J. Artif. Intell. Res.(JAIR)*, vol. 32, pp. 565–606, 2008.

[12] P. O. Stalph, M. Ebner, M. Michel, B. Pfaff, and R. Benz, in *GECCO*, C. Ryan and M. Keijzer, Eds., pp. 535–536.

[13] S. Teraoka, T. Ushio, and T. Kanazawa, "Voronoi coverage control with time-driven communication for mobile sensing networks with obstacles." in *CDC-ECE*. IEEE, 2011, pp. 1980–1985. [Online]. Available: http://dblp.uni-trier.de/db/conf/cdc/cdc2011.html#TeraokaUK11

[14] O. Lejri and M. Tagina, "Representation in case-based reasoning applied to control reconfiguration," in *Advances in Data Mining. Applications and Theoretical Aspects*, ser. Lecture Notes in Computer Science, P. Perner, Ed. Springer Berlin Heidelberg, 2012, vol. 7377, pp. 113–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31488-9_10

[15] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in *17th International Conference on Principles and Practice of Constraint Programming*, 2011, pp. 454–469.

[16] M. Gagliolo and J. Schmidhuber, "Learning dynamic algorithm portfolios," vol. 47, no. 3-4, 2006, pp. 295–328.

[17] W. Armstrong, P. Christen, E. McCreath, and A. P. Rendell, "Dynamic algorithm selection using reinforcement learning," in *International Workshop on Integrating AI and Data Mining*, 2006, pp. 18–25.

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

[19] K. Gale and Tucker, "Linear programming and the theory of games," in *Activity Analysis of Production and Allocation*, Koopmans, Ed. Wiley, 1951, ch. XII.

[20] 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.

[21] 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.

[22] J. Paredis, "Coevolutionary computation," *Artificial Life*, vol. 2, pp. 355–375, 1995.

[23] C. Ong, H. Quek, K. Tan, and A. Tay, "Discovering chinese chess strategies through coevolutionary approaches," in *IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 360–367.

[24] P. Drake and Y.-P. Chen, "Coevolving partial strategies for the game of go," in *International Conference on Genetic and Evolutionary Methods*. CSREA Press, 2008.

[25] H. Samulowitz and R. Memisevic, "Learning to solve qbf," in *Proceedings of the 22nd National Conference on Artificial Intelligence*. AAAI, 2007, pp. 255–260.

[26] T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, pp. 4–22, 1985.

[27] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2/3, pp. 235–256, 2002.

[28] J. Audibert, R. Munos, and C. Szepesvri, "Exploration-exploitation trade-off using variance estimates in multi-armed bandits," *Theoretical Computer Science*, 2008.

[29] 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.

[30] B. Robertie, "Backgammon," *Inside Backgammon*, vol. 2, no. 1, p. 4, 1980.