Online Sparse bandit for Card Games

David Saint-Pierre, Quentin Louveaux, Olivier Teytaud

Online Sparse bandit for Card Games

David L. St-Pierre[1], Quentin Louveaux[1], and Olivier Teytaud[2,3]

[1] Department of Electrical Engineering and Computer Science, Faculty of Engineering, Liège University, Belgium
[2] TAO (Inria, Lri, Univ. Paris-Sud, UMR CNRS 8623), France
[3] OASE Lab, National University of Tainan, Taiwan

Abstract. Finding an approximation of a Nash equilibria in matrix games is an important topic that reaches beyond the strict application to matrix games. A bandit algorithm commonly used to approximate a Nash equilibrium is EXP3 [**?**]. However, the solution to many problems is often sparse, yet EXP3 inherently fails to exploit this property. To the knowledge of the authors, there exist only an offline truncation proposed by [**?**] to tackle such issue. In this paper, we propose a variation of EXP3 to exploit the fact that solution is sparse by dynamically removing arms; the resulting algorithm empirically performs better than previous versions. We apply the resulting algorithm to a MCTS program for the Urban Rivals card game.

1 Introduction

Bandits algorithms [**?**,**?**,**?**,**?**] are tools for handling the dilemma of exploration vs exploitation. In particular, they are useful for finding Nash equilibria of matrix games [**?**,**?**]. Finding Nash equilibria of matrix games is an important topic; beyond its strict application, which models many financial applications, matrix games are a component of many games, even those represented by a set of rules. It has been shown in [**?**] how matrix games can be used inside a Monte-Carlo Tree Search implementation, for extending Monte-Carlo Tree Search to games with simultaneous actions, and [**?**] has extended this to games with partial information.

It is possible to use Linear Programming (LP) to compute an exact Nash Equilibrium (NE) of a zero-sum matrix game; however, it is not possible to apply LP for solving huge matrix games as players commonly encouter. Indeed, in many cases of interest (e.g. when used inside a Monte-Carlo Tree Search implementation), we want to minimize the number of accesses to the matrix, because the cost of solving the game lies more in the computation of the numbers in the matrix than in the solving itself. We want algorithms which find an approximate solution without reading the entire matrix. As [**?**] shows, one can find an ϵ-approximate Nash equilibrium of a zero-sum $K \times K$ matrix game by accessing only $O(K \log(K)/\epsilon^2)$ elements of the matrix, i.e. by far less than the $K \times K$ elements of the matrix. As shown in [**?**], EXP3 and INF, an EXP3-like variant to address the adversarial multiarmed bandit problem, have the same property;

moreover, they can be applied in a more general setting, namely, when each element of the matrix is only known up to a finite number of measurements, the average of which converges to the real value. As a consequence, bandit tools are now known as a standard solution for approximate solving of matrix games. Counterfactual regret[**?**] is another possible approach, yet it is for the moment devoted to extensive form games rather than matrix games.

However, a crucial element in games has not been used in these works: the fact that, typically, the solutions found in games are sparse. In other words, a game can contain a lot of choices, yet only a very few among them are interesting.

This paper focuses on problems where the computation via LP of a NE is too intensive and therefore needs to be approximated. Furthermore, we seek an algorithm that does not require visiting each element of a potentially very large matrix. Based on this premise, to the best of our knowledge, the only paper in which sparsity in Nash equilibria was used for accelerating bandits is [**?**], with the offline algorithm discussed in section **??**; we here investigate improved versions, working online.

Section **??** presents the framework of sparse bandits and related algorithms. Section **??** presents experimental results on artificial and real world datasets.

2 Algorithms for sparse bandits

Sparse bandits are bandit problems in which the optimal solution is sparse; in the case of bandits for approximating Nash equilibria, this means that the Nash equilibria use only a small number of components, i.e. if x^*, y^* is a Nash equilibrium and the matrix has size $K \times K$, then $\{i; x^*_i > 0\}$ and $\{i; y^*_i > 0\}$ both have cardinal $<< K$.

We present below the framework of Nash equilibria in matrix games (section **??**). We then present our version of the EXP3 algorithm (section **??**), and then our modified versions for sparse problems (section **??**).

2.1 Matrix games and Nash equilibria

Consider a matrix M of size $K \times K$ with values in $[0, 1]$ (we choose a square matrix for short notations, but the extension is straightforward). Player 1 chooses an action $i \in [[1, K]]$ and player 2 chooses an action $j \in [[1, K]]$; both actions are chosen simultaneously. Then, player 1 gets reward $M_{i,j}$ and player 2 gets reward $1 - M_{i,j}$; the game therefore sums to 1. We consider games summing to 1 for commodity of notations in EXP3, but 0-sum games are obviously equivalent. A Nash equilibrium of the game is a pair (x^*, y^*) (both in $[0, 1]^K$ and summing to 1) such that if i is distributed according to the distribution x^* (i.e. $i = k$ with probability x^*_k) and if j is distributed according to the distribution y^* (i.e. $j = k$ with probability y^*_k) then neither player can expect a better average reward by unilaterally changing its strategy.

2.2 EXP3 algorithm

We use a version of EXP3[**?**] inspired from [**?**]. EXP3 is an algorithm for bandits problems; here, we use two EXP3 bandits simultaneously: one for the row player and one for the column player. This is a tool for adversarial bandits, including matrix games.

Our version of EXP3 is explained in Alg. **??**. Alg. **??** would be run for each player independently.

Algorithm 1 EXP3 algorithm for iteration t with K arms.

Initialise $\forall i, p(i) = \frac{1}{K}, n(i) = 0, S(i) = 0$; $t = 0$
while t < T **do**
  Arm i is chosen with probability $p(i)$
  n(i) ← n(i)+1
  Receive reward r
  t ← t+1
  S_i modified by the update formula $S_i \leftarrow S_i + r/p(i)$ (and S_j for $j \neq i$ is not modified).
  $\forall i, p(i) = 1/(K\sqrt{t}) + (1 - 1/\sqrt{t}) \times \exp(S_i/\sqrt{t})/\sum_j \exp(S_j/\sqrt{t})$
end while
return n

The ratio between the number of times an arm was pulled and the total number of iterations converges to the Nash equilibrium as explained in [**?**].

2.3 Sparse EXP3 bandits

In the subsections below, we introduce a truncated EXP3 (section **??**) and introduce our online pruning versions (section **??**).

Truncated EXP3. [**?**] proposed to modify the EXP3 algorithm to better exploit the sparsity in the solution, as explained in Alg. **??**. The first step is to run the EXP3 algorithm, providing an approximation (x, y) of the Nash equilibrium and the second step executes a truncation following Alg. **??**. This truncation uses the property that, over time, the probability to pull an arm that is not part of the optimal solution will tend toward 0. Therefore, as they are likely to be outside the optimal solution, it artificially truncates the arms which have a low probability to be pulled, based upon a threshold c.

The constant c is chosen as $\max_i(Tx_i)^\alpha/T$ for some $\alpha \in]0, 1[$ (and d accordingly), as in [**?**], and T is the number of iterations of the EXP3 algorithm. $\alpha = 0.8$ is proposed in [**?**].

onEXP3. The question investigated in this paper is whether it is possible to dynamically truncate some arms to improve performance. In its current form, the

Algorithm 2 TEXP3, the truncation after EXP3.

Let x and y be the approximate Nash equilibria as proposed by EXP3 for the row and column players respectively.
Truncate as follows:

$$x'_i = x_i \text{ if } x_i > c, x'_i = 0 \text{ otherwise;}$$
$$y'_i = y_i \text{ if } y_i > d, y'_i = 0 \text{ otherwise.}$$

Renormalize: $x'' = x'/\sum_i x'_i$; $y'' = y'/\sum_i y'_i$.
Output x'', y''.

online EXP3 algorithm that we propose assumes the total number of iterations T is known. This assumption is relatively common in games as it can be viewed as an upper bound on the time allowed to take a decision.

Basically, onEXP3 extends TEXP3 by gradually increasing a threshold c_t. The first version of onEXP3 is solely based on the number of iterations t to determine c_t. With the knowledge of T, it is easy to extrapolate a fixed c_T and let c_t tend toward c_T.

It is important to bear in mind that it is the ratio between the number of times a specific arm x_i is pulled over the total number of iterations T that converges toward a Nash, not the probability $p(i)$ given in EXP3. Therefore, the decision on whether to prune is based upon the number of times the arm was pulled. The main pitfall of using this method is to remove an arm that was not explored enough. To prevent this situation, a lower threshold was included. Any arm pulled less than this threshold cannot be removed. In the current version, we used $x_i > \lceil \frac{t}{K} \rceil$.

The point is to remove every arm that, based on the current information, will not fulfill the final condition $x_i > c_T$ (and $y_j > d_T$). We used $x_i < (b_1 \times T^\delta \times \left(\frac{t}{T}\right)^\beta)$ where t is the current number of iterations, T is the maximum number of iterations, b_1 and $\beta \in \mathbb{R}$ and $\delta \in]-1, 0[$ (typically $\delta = \alpha - 1$ and $\beta = 3$). In the worst case scenario, i.e., if every arm were pruned, we turn back to normal EXP3. To sum up, Algorithm **??** presents the modified EXP3 (for one player; the same algorithm is applied for the other player).

onEXP3v2 Algorithm **??** is built upon the assumption that the matrices are square. For rectangular matrices, let us assume for simplicity that $k_1 > k_2$ (k_1 number of rows, k_2 number of columns). In onEXP3v2, we assume that the cut depends not on the number of iterations, but rather on the number of arms. Therefore, the number of arms must be included in the dynamic truncation condition. We chose to use $max_i(x_i)^\alpha$ as the truncating factor.

Thus, $x_i < (b_1 \times T^\delta \times \left(\frac{t}{T}\right)^\beta)$ becomes $x_i < (b_2 \times max_i(x_i)^\alpha \times \left(\frac{t}{T}\right)^\beta)$ where $b_2 \in \mathbb{R}$. To mitigate the relative difference in terms of the number of times an arm

Algorithm 3 onEXP3, an online EXP3 algorithm with a cut solely based on T.

Initialise $\forall i, p(i) = \frac{1}{K}, n(i) = 0, S(i) = 0$; $t = 0$
while t < T **do**
  Arm i is chosen with probability $p(i)$
  n(i) ← n(i)+1
  t ← t+1
  Receive reward r
  S_i modified by the update formula $S_i \leftarrow S_i + r/p(i)$
  $\forall i, p(i) = 1/(K\sqrt{t}) + (1 - 1/\sqrt{t}) \times \exp(S_i/\sqrt{t})/\sum_j \exp(S_j/\sqrt{t})$
  if $x_i > \lceil \frac{t}{K} \rceil$ **and** $x_i < (b_1 \times T^{\delta} \times \left(\frac{t}{T}\right)^{\beta})$ **then**
    Remove arm i
  end if
  if every arm has been pruned **then**
    Use plain EXP3
  end if
  Renormalize: $p = p/\sum_i p(i)$
end while
Execute the truncation TEXP3 as presented in **??**
return n

was pulled between the two players, a factor $\frac{k_1}{k_2}$ is added in front of the segment $\lceil \frac{t}{K} \rceil$. To sum up, Algorithm **??** presents the modified online EXP3 (onEXP3v2).

3 Computational results

This section presents the computational results. Section **??** shows the results on randomly generated deterministic matrix games. Section **??** presents a version on non-square matrix games. Section **??** describes the results on Urban Rival, a simultaneous stochastic rectangular matrix game.

To generate a matrix that gives a sparse Nash equilibrium, we create a method that takes the number of arms k and a sparsity parameter γ. Then, a square submatrix of size k' is filled with $\{0, 1\}$ uniformly distributed with equal probabilities, where $k' = k(1-\gamma)$ represents a small number that depends on γ. The $k - k'$ rows are filled with 0 with a probability of $1 - \gamma$ and 1 with a probability of γ (and the opposite for the columns). To ensure that the algorithms do not exploit a form of smoothness in the solution, we shuffle the rows and the columns. This protocol is trivially extendable to rectangular matrices.

Table **??** and Table **??** show the results of games in different settings. The column 'vs' displays the opponents. The column 'γ' gives information on the relative sparsity of the solution the higher the number, the sparser the solution is. Empirical experiments showed that at 0.8 one can expect $0.3k$ arms in the solution, at 0.9 around $0.1k$ arms and at 0.99 approximately $0.05k$ arms. The columns starting with '$\frac{T}{k1 \times k2}$' followed by a number state the relative number of iterations executed to reach these results, given in percentage.

Algorithm 4 onEXP3v2, an online EXP3 algorithm with a cut based on k.

Initialise $\forall i, p(i) = \frac{1}{K}, n(i) = 0, S(i) = 0$; $t = 0$
while t <T **do**
  Arm i is chosen with probability $p(i)$
  n(i) $\leftarrow$ n(i)+1
  t $\leftarrow$ t+1
  Receive reward r
  S_i modified by the update formula $S_i \leftarrow S_i + r/p(i)$
  if $x_i > \frac{k_1}{k_2} \times \lceil \frac{t}{K} \rceil$ **and** $x_i < (b_2 \times max_i(x_i)^{\alpha} \times \left(\frac{t}{T}\right)^{\beta})$ **then**
    remove arm i
  end if
  $\forall i$ not discarded, $p(i) = 1/(K\sqrt{t}) + (1 - 1/\sqrt{t}) \times \exp(S_i/\sqrt{t}) / \sum_j \exp(S_j/\sqrt{t})$
  Renormalize: $p = p/\sum_i p(i)$
end while
Execute the truncation TEXP3 as presented in **??**
return n

As each run of one of our algorithm provides both a strategy for the row player and a strategy for the column player. we can compare algorithm for both sides. The score that we use is therefore equal to the mean of its performance when it plays as the row player combined with when it plays as the column player. This measure is more reliable than the distance to the optimal solution because a near-optimal solution can be weak in general against other suboptimal players. Thus, when an algorithm is playing against the optimal solution it can never reach more than 0.5. Also, the Nash equilibrium is not always possible to compute in games, therefore a strong performance against other algorithms is at least as important as performing well against the best solution. In our case, we look for an algorithm that performs well against both Nash Equilibria and suboptimal players. Every score is given in percentage.

3.1 Generated squared matrix games

Every experiment was conducted over 100 randomly generated matrices. The seed for the random selection was fixed to reduce variance (noise) in the outcomes.

In Table **??**, onEXP3 clearly outperforms TEXP3 when $\frac{T}{k1 \times k2}$ is between 5% and 50% of the number of elements in the matrix when playing with 200 arms on each side. When the matrices have a size of 100 arms, results are still strong. A caveat, at 50 arms and a very low number of iterations, the performance is lower in only one setting 49% (at 0.05) in which onEXP3 is weaker than TEXP3. When the number of iterations increases (0.1, 0.25, 0.5), the score (53%, 52%, 54%) is well over 50%. This suggests a better performance from onEXP3 for greater numbers of arms. As for the sparsity (represented by γ), even though it makes the scores change from 51%, 53%, 62%, 56% at 0.8 to 57%, 53%, 56%, 53% at 0.99, it does not change the general domination of onEXP3 over TEXP3

vs	γ	$k_1 \times k_2$	$\frac{T}{k1\times k2} = 5\%$	$\frac{T}{k1\times k2} = 10\%$	$\frac{T}{k1\times k2} = 25\%$	$\frac{T}{k1\times k2} = 50\%$
onEXP3 vs TEXP3	0.9	200 × 200	54 ± 1.14	58 ± 1.03	57 ± 0.91	56 ± 2.42
onEXP3 vs TEXP3	0.9	100 × 100	54 ± 1.13	53 ± 0.91	54 ± 0.90	58 ± 2.37
onEXP3 vs TEXP3	0.8	100 × 100	51 ± 0.55	53 ± 1.01	62 ± 1.21	56 ± 2.24
onEXP3 vs TEXP3	0.99	100 × 100	57 ± 1.70	53 ± 0.82	56 ± 1.87	53 ± 1.99
onEXP3 vs TEXP3	0.9	50 × 50	49 ± 0.12	53 ± 1.02	52 ± 0.77	54 ± 1.99
onEXP3 vs EXP3	0.9	200 × 200	78 ± 1.08	80 ± 0.17	80 ± 0.34	80 ± 0.32
onEXP3 vs EXP3	0.9	100 × 100	76 ± 1.21	80 ± 0.22	79 ± 0.27	78 ± 0.39
onEXP3 vs EXP3	0.8	100 × 100	67 ± 0.87	68 ± 0.40	68 ± 0.30	69 ± 0.46
onEXP3 vs EXP3	0.99	100 × 100	88 ± 1.35	**92 ± 0.09**	90 ± 0.28	88 ± 0.37
onEXP3 vs EXP3	0.9	50 × 50	75 ± 1.16	80 ± 0.28	78 ± 0.50	78 ± 0.42
onEXP3 vs NE	0.9	200 × 200	29 ± 0.54	30 ± 0.53	37 ± 0.81	43 ± 0.65
onEXP3 vs NE	0.9	100 × 100	19 ± 0.66	31 ± 0.64	35 ± 0.60	40 ± 0.75
onEXP3 vs NE	0.8	100 × 100	28 ± 0.33	37 ± 0.38	38 ± 0.45	42 ± 0.55
onEXP3 vs NE	0.99	100 × 100	22 ± 1.67	38 ± 0.92	40 ± 0.96	42 ± 0.90
onEXP3 vs NE	0.9	50 × 50	18 ± 0.65	20 ± 0.79	33 ± 0.70	38 ± 0.82
TEXP3 vs NE	0.9	200 × 200	15 ± 0.19	17 ± 0.33	25 ± 0.91	38 ± 0.94
TEXP3 vs NE	0.9	100 × 100	17 ± 0.37	19 ± 0.47	28 ± 0.93	35 ± 0.95
TEXP3 vs NE	0.8	100 × 100	27 ± 0.20	28 ± 0.22	32 ± 0.57	38 ± 0.68
TEXP3 vs NE	0.99	100 × 100	19 ± 1.56	28 ± 1.61	38 ± 0.94	40 ± 0.98
TEXP3 vs NE	0.9	50 × 50	19 ± 0.65	19 ± 0.58	28 ± 0.99	35 ± 0.92

Table 1. Performance of onEXP3. $\beta = 3$ in all experiments.

for these settings. The general decrease in the score between 0.25 and 0.5 can be explained by the fact that TEXP3 makes better decisions as the amount of available information increases.

When onEXP3 plays against EXP3, the results are clear: onEXP3 performs much better. Obviously, since it is the purpose of the algorithm it is not surprising. The variation in the size of the matrices changes the relative score, yet it does not change the outcome: onEXP3 is stronger. The variation in the sparsity yields the same conclusion; The score is consistent with the previous finding.

OnEXP3 vs NE and TEXP3 vs NE give an insight on their relative performance against the best possible player. Aside from the lowest arms setting (50 arms), onEXP3 is always higher than TEXP3, sometimes the difference being as much as 14% (31%-17% at 200 arms). From the result, we can infer that onEXP3 tends faster toward a good solution. However, because it prunes dynamically, it has no guarantee of convergence (albeit converges with most of our randomly generated matrices). It performs consistently better than TEXP3 against the Nash equilibrium.

Therefore, within the setting presented in Table **??**, it appears that onEXP3 is better than TEXP3 against both an optimal player and a suboptimal one.

Overall, the more information each algorithm receives, the better it plays; From $\frac{T}{k1\times k2}$ equal to 0.05 up to 0.5, each algorithm improved its play against the Nash equilibrium. It seems that at $\frac{T}{k1\times k2} = 5$, onEXP3 does not have enough information to truncate thus explaining the relatively lower performance (still significantly better) against TEXP3 when compared to results with higher num-

vs	γ	$k_1 \times k_2$	$\frac{T}{k1\times k2} = 5\%$	$\frac{T}{k1\times k2} = 10\%$	$\frac{T}{k1\times k2} = 25\%$	$\frac{T}{k1\times k2} = 50\%$
onEXP3v2 vs TEXP3	0.9	400 × 40	52 ± 0.49	53 ± 0.37	55 ± 0.35	56 ± 0.33
onEXP3v2 vs TEXP3	0.9	200 × 20	51 ± 0.53	52 ± 0.51	52 ± 0.31	55 ± 0.29
onEXP3v2 vs TEXP3	0.9	100 × 10	52 ± 0.53	51 ± 0.54	54 ± 0.54	54 ± 0.39
onEXP3v2 vs TEXP3	0.8	200 × 20	51 ± 0.53	52 ± 0.54	55 ± 0.43	55 ± 0.34
onEXP3v2 vs TEXP3	0.99	200 × 20	51 ± 0.58	54 ± 0.63	52 ± 0.27	54 ± 0.29
onEXP3v2 vs TEXP3	0.9	200 × 50	52 ± 0.54	53 ± 0.37	54 ± 0.32	54 ± 0.25
onEXP3v2 vs EXP3	0.9	400 × 40	83 ± 0.28	84 ± 0.15	84 ± 0.14	83 ± 0.14
onEXP3v2 vs EXP3	0.9	200 × 20	67 ± 0.62	78 ± 0.42	84 ± 0.17	84 ± 0.16
onEXP3v2 vs EXP3	0.9	100 × 10	58 ± 0.62	62 ± 0.61	78 ± 0.50	83 ± 0.28
onEXP3v2 vs EXP3	0.8	200 × 20	61 ± 0.47	69 ± 0.40	74 ± 0.24	74 ± 0.20
onEXP3v2 vs EXP3	0.99	200 × 20	67 ± 0.83	87 ± 0.72	**94 ± 0.23**	93 ± 0.13
onEXP3v2 vs EXP3	0.9	200 × 50	79 ± 0.46	82 ± 0.23	82 ± 0.41	81 ± 0.13
onEXP3v2 vs NE	0.9	400 × 40	37 ± 0.39	39 ± 0.33	41 ± 0.27	42 ± 0.22
onEXP3v2 vs NE	0.9	200 × 20	26 ± 0.61	37 ± 0.42	43 ± 0.24	44 ± 0.19
onEXP3v2 vs NE	0.9	100 × 10	17 ± 0.69	22 ± 0.69	39 ± 0.58	45 ± 0.33
onEXP3v2 vs NE	0.8	200 × 20	31 ± 0.46	37 ± 0.42	42 ± 0.29	44 ± 0.23
onEXP3v2 vs NE	0.99	200 × 20	20 ± 0.86	41 ± 0.73	48 ± 0.23	48 ± 0.15
onEXP3v2 vs NE	0.9	200 × 50	33 ± 0.58	36 ± 0.48	37 ± 0.31	40 ± 0.31
TEXP3 vs NE	0.9	400 × 40	35 ± 0.37	38 ± 0.28	39 ± 0.24	41 ± 0.18
TEXP3 vs NE	0.9	200 × 20	25 ± 0.59	35 ± 0.45	42 ± 0.23	43 ± 0.17
TEXP3 vs NE	0.9	100 × 10	14 ± 0.66	20 ± 0.65	34 ± 0.57	42 ± 0.36
TEXP3 vs NE	0.8	200 × 20	30 ± 0.43	36 ± 0.40	40 ± 0.27	43 ± 0.20
TEXP3 vs NE	0.99	200 × 20	20 ± 0.79	37 ± 0.73	48 ± 0.19	48 ± 0.17
TEXP3 vs NE	0.9	200 × 50	33 ± 0.58	35 ± 0.23	37 ± 0.40	39 ± 0.29

Table 2. Performance of onEXP3v2 . $\beta = 3$ in all experiments.

ber of iterations. The best performances are with a higher number of arms which only means the tuning is more efficient for this size of matrices.

Let us look at the trends of performance between the size of the matrices: In the setting 50×50 the results, while not as strong as in 100×100, still show a better performance from onEXP3. When the matrices are 200×200, results are impressive achieving a $92 \pm 0.09\%$ with few iterations ($\frac{T}{k1\times k2} = 10$), suggesting that larger matrices yield better performance.

When comparing onEXP3 and TEXP3 against the Nash equilibrium over a γ of 0.9 and 0.8, both improve their respective performance as sparsity gets lower. It is not really surprising considering that the real difficulty is to prune arms that are not promising. There is an observable peak in terms of performance of onEXP3 against TEXP3 in a high sparsity setting (0.99). OnEXP3 truncates within the EXP3 iteration allowing to focus more rapidly on the few good arms thus explaining this peak.

3.2 Generated general matrix games

Each experiment was conducted over 500 randomly generated matrices. The seed for the random selection was fixed to reduce variance (noise) in the outcomes.

Table **??** presents the results achieved by the algorithms in a rectangular matrix setting. OnEXP3v2 outperforms TEXP3 when $\frac{T}{k1\times k2}$ is between 0.05 and 0.5 percent of the number of elements in the matrix when playing with

400×40 arms. When the matrices have a size of 200×20 arms, the results are still strong. At 100×10 arms the score is well over 50%. This suggests a better performance from onEXP3v2 when the number of arms augments. The sparsity has an impact on the score, yet does not change the general domination of onEXP3v2 over TEXP3 for these settings.

When onEXP3v2 is playing against EXP3, the results are clear: onEXP3v2 performs much better, with results similar to Table **??**. Obviously, since the purpose of the algorithm is to lean faster towards a better solution, it is not surprising. The variation in the size of the matrices changes the relative score, yet it does not change the outcome: onEXP3v2 is stronger. The variation in the sparsity yields the same conclusion; The score is consistent with the previous finding.

The difference between the performance of onEXP3v2 and TEXP3 against the best possible player, while not as strong as in Table **??**, still gives the edge to onEXP3v2. Again it seems that larger matrices yield better performance. OnEXP3v2 statistically beats TEXP3 with a 95% confidence interval in most cases. Also, there was no setting in which the mean was under 50%. When onEXP3v2 plays against EXP3, the domination is statistically observable at an interval of 3 times the standard deviation, even managing a peak at $94 \pm 0.23\%$. At a lower sparsity (e.g. 0.8), onEXP3v2 does not perform as well as its predecessor onEXP3. Nevertheless, it still manages good performance overall, but the sparsity does not impact as much as in Table **??**.

It seems the narrow side of the matrix limits the effectiveness of the sparsity pruning. Less extreme rectangular matrix, such as 200×50, demonstrates this conclusion by having values higher than 200×20 and lower than 200×200. A very interesting result is the impact of the amount of information on the score. The score does not change as drastically as it did in Table **??**.

3.3 Application to UrbanRivals

For the sake of comparison, we applied onExp3v2 to the same card game as [**?**], namely Urban Rivals, thanks to the code kindly provided by the authors. We implemented onExp3 and made it play against the version of TEXP3 used in [**?**].

Urban Rivals (UR) is a widely played internet card game, with partial information. As pointed out in [**?**], UR can be consistently solved by a Monte-Carlo Tree Search algorithm (MCTS) thanks to the fact that the hidden information is frequently revealed: a sequence of time steps without revealed information are integrated into one single matrix game, so that the game is rephrased as a tree of matrix games with finite horizon. EXP3 is used in each node (therefore EXP3 is used for solving many matrix games), each reading of a coefficient in the payoff matrix at depth d of the tree being a simulated game (by MCTS itself) with depth $d-1$. As a consequence, reading coefficients in the payoff matrices at the root is quite expensive, and we have to solve the game approximately. We refer to [**?**] for more details on the solving of partial information game with periodically revealed information by MCTS with EXP3 bandit.

The overall algorithm is a Monte-Carlo Tree Search, with EXP3 bandits in nodes; the sparse bandit is applied at the root node of the algorithm. We compared results for a similar number of iterations and obtained success rates as presented in Table **??**.

The main difficulty the algorithm faces in this application is the evaluation of the reward. When EXP3 is used in a MCTS settings, results are stochastic (in the sense that playing a same i and j for the row and column player respectively does not always lead to the same result). OnExp3v2 relies on the assumption that the reward is fixed to execute a more aggressive pruning, which is not the case in this application. Yet with little tuning, onEXP3v2 significantly outperforms TEXP3 when the number of simulations reaches 1600 and 3200.

The drop in the score when the simulations reach 6400 and 12800 is consistent with the findings in Table **??** and Table **??**. There is enough information gathered during the iteration phase to adequatly prune offline. As such, these values are in fact showing that onEXP3v2 remains as good as TEXP3 when the number of iterations is high.

# simulations per move	score $\pm\sigma$
100	0.519 ± 0.014
200	0.509 ± 0.014
400	0.502 ± 0.014
800	0.494 ± 0.015
1600	0.558 ± 0.014
3200	0.549 ± 0.014
6400	0.494 ± 0.015
12800	0.504 ± 0.015

Table 3. Results of onExp3 for UrbanRivals (against the version of TEXP3 developed by the authors of [**?**]). The improvement is moderate but significant; we point out that the game is intrinsically noisy, so that 55% is already a significant improvement as only the average of many games make sense for evaluating the level of a player.

4 Conclusion

EXP3 and related algorithms are great tools for computing approximate Nash equilibria. However, they do not benefit from sparsity. There already exists versions of EXP3 for sparse cases, but these versions only benefit from an offline sparsity; EXP3 is run, and, thereafter, some arms are pruned. This paper proposes online pruning algorithms (onEXP3 and onEXP3v2 for square and rectangular matrices respectively), dynamically removing arms, with a variable benefit (from minor to huge depending on the framework), and in all cases a significantly non-negative result.

The benefit on the real-world game Urban Rivals is moderate but significant. Our online sparsity algorithm scales well as it becomes more and more efficient as the game becomes bigger.

The main further works are as follows. First, whereas TEXP3 (from [**?**]) makes no sense in internal nodes of a MCTS tree, our modified (online) version makes sense for all nodes - therefore, we might extend our application to Urban Rivals by applying online sparsity to all nodes, and not only at the root. This is a simple modification which might lead to big improvements.

The second further work is a formal analysis of onEXP3. Even TEXP3 only has small theoretical analysis; the algorithms are so stable and so scalable that we believe a mathematical analysis is possible.

The third further work is to compare a framework based on the combination of MCTS with onEXP3 to other similar opponents such as MCCFR [**?**] and MCRNR [**?**].

Acknowledgements

This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors.