Minimax Strikes Back: Technical Appendix

1 Introduction

This document describes the details of experiments of the paper, removed for a reason of space. In particular, we describe in this appendix: the used games (Section 2.1), the used computational ressources (Section 2.2), the used neural network architectures and the learning parameters of experiments. Some details about results are also provided, such that the evolution of win rate curves for Descent and Polygames, in Section 2.4 and about the learning data analysis, in Section 2.5. In Section 2.6, neural networks architectures and parameters of Sections 2.7 and 2.8 are exposed. In Section 2.7, other long training processes have been carried for comparing Descent and Polygames, leading to results analogous to the main article. Details about comparison against tournament Polygames networks are provided in Section 2.8. Finally, in Section 2.9, we provide answers to questions from reviewers of previous submissions.

2 Technical Details for the Paper Experiments

2.1 Tested Games

In this section, we present the games on which we compare the algorithms. Note that, on the one hand, these games are recurring at the Computer Olympiad, the world multi-game competition for board game programs. On the other hand, these games are available in the Ludii games library [Piette *et al.*, 2020].

Hex

The game of Hex is a connection game, with very simple rules. Players place one piece of their color per turn and aim to connect the two board sides of their color. Despite this apparent simplicity, the Hex strategies are very complex, the number of states is greater than Chess (from board size 11×11), and the complexity of the game is PSPACE-complete [Bonnet *et al.*, 2016].

Othello

The Othello game (also known as Reversi) is a territory game. In their turn, players place a piece of their color in a cell where they can encircle opponent's pieces with another of their pieces, already placed. The encircled opponent pieces are then replaced by pieces of their color. The winner is the one who has the most pieces of their color on the board. The Othello is also PSPACE-complete [Iwata and Kasai, 1994].

Breakthrough

Breakthrough is a race game. It is played with the pawns of the Chess game. The objective of the players is to be the first to reach the opposite side with one of their pawns.

Each of these three games can be played on a varying size board. The standard size at Othello and at Breakthrough is 8×8. In Hex, 11×11, 13×13 and 19×19 are the standard sizes.

Surakarta

Surakarta is played on an original board of size 6×6. It belongs to the game family of international draughts. Pieces have two types of movement: a simple movement to an adjacent square and a capture movement. The objective is to capture all the opposing pieces. The differences with the classic game of draughts is that the simple movement can be carried out towards any adjacent square (orthogonal or diagonal) and that the capture is carried out when arriving on the opposing piece (and not by jumping over) and must follow a specific movement, which consists of traversing one or more times specific movement circuits specific to the captures.

Outer-Open-Gomoku

Outer-Open-Gomoku is an alignment game. At each turn, each player put a stone on the 15×15 square board. The winner is the first which aligns 5 of their stones in any direction.

Connect6

Connect6 is analogous to Outer-Open-Gomoku except that the board game is 19×19 and that players put two stones at their turn (only one for the first player at their first turn to mitigate their advantage) with the goal of aligning 6 stones. Connect6 is PSPACE-complete [Hsieh and Tsai, 2007].

Havannah

Havannah is similar to Hex, i.e. it is a positional connection game. The goal is to be the first to connect 3 edges of the board with stones of their color or 2 board corners with stones of their color, or to create a loop on the board with stones of their color. The board is a regular hexagonal board with 6 corners. Havannah is PSPACE-complete [Bonnet *et al.*, 2013].

2.2 Computational Resources

In this section, we present the used computational resources.

For the performed training runs and the performed confrontations, we use the following hardware: GPU Nvidia Tesla V100 SXM2 32 Go, 2 to 10 CPU (processors Intel Cascade Lake 6248 2.5GHz) on RedHat. There is an exception, for the performed confrontations against Polygames tournament networks, we use the following hardware: GeForce GTX 1080 Ti, 2 to 8 CPU (Intel(R) Xeon(R) CPU E5-2603 v3 1.60GHz) on Ubuntu 18.04.5 LTS.

Descent programs (Descent learning, Unbounded Minimax, ...) are coded in Python (using tensorflow 1.15). Games and Search in Polygames are coded in C/C++. For confrontations, Polygames num_actor parameter is 8 (threads doing MCTS).

2.3 Algorithms of the Two Learning Framework

In this section, we briefly present some of the algorithms of the two learning framework.

The learning process with the Descent framework is as follows. As long as there is time left, a new match phase is performed. A match phase consists of a match against oneself, where in each turn the move to be played is decided after carrying out a search with Descent. The move to be played after the search is chosen according to an action selection method, depending on the result of the search. In these experiments, the action selection method used is the *ordinal law* [Cohen-Solal, 2020] with the exploitation parameter ϵ' choosing at random uniformly between 0 and 1, each time a new action must be decided . After each match phase, the data from the associated partial game tree is added to the previous data (only the data of the last 100 matches are kept) and a training phase is carried out from a sample of this data set.Specifically, smooth experience replay is used [Cohen-Solal, 2020]. The main part of this algorithm is described in Algorithm 1. The full formalization is described in [Cohen-Solal, 2020].

Function `Descent_main_algorithm`($t_{\max}$)
  $t_0 \leftarrow$ `time()`
  while `time()` $- t_0 < t_{\max}$ **do**
    $s \leftarrow$`initial_game_state()`
    $S \leftarrow \emptyset$
    $T \leftarrow \{\}$
    while $\neg$`terminal(`s`)` **do**
      $S, T \leftarrow$`descent(`$s, S, T, f_\theta, f_{\mathrm{t}}$`)`
      $a \leftarrow$`action_selection(`s, S, T`)`
      $s \leftarrow a(s)$
    $D \leftarrow \{(s, v(s)) \mid s \in S\}$
    `update(`f_θ, D`)`

Algorithm 1: Main algorithm of the Descent framework (see Table 1 for the definitions of symbols). In this context, S is the set of states which are non-leaves or terminal.

The learning process with AlphaZero [Silver *et al.*, 2018] is as follows. As long as there is time left, a new match phase is performed. A phase consists of a match against oneself, where in each turn the move to be played is decided after carrying out a search with MCTS + PUCT. The algorithm of MCTS is similar to Unbounded Minimax, The first main differences is that the value of a state is not the minimax value in the partial game tree but the average of the leaves in the subtree starting from that state. The second main difference is that the tree is constructed not in choosing states of higher value, but states optimizing the value plus an exploration term depending on the policy of the neural network and the number of selection of actions during the research. After the match, the match state sequence data is added to the previous data (only the most recent data is kept). Every K matches (K is a parameter), training is performed from a sample of this data set. The main part of this algorithm is described in Algorithm 2.

Function `AlphaZero_main_algorithm`($t_{\max}$)
  $t_0 \leftarrow$ `time()`
  while `time()` $- t_0 < t_{\max}$ **do**
    for $k \in \{1, \ldots, K\}$ **do**
      $s \leftarrow$`initial_game_state()`
      $S \leftarrow \emptyset$
      $T \leftarrow \{\}$
      $G \leftarrow \{s\}$
      while $\neg$`terminal(`s`)` **do**
        $S, T \leftarrow$`mcts(`$s, S, T, f_\theta, f_{\mathrm{t}}$`)`
        $a \leftarrow$`action_selection(`s, S, T`)`
        $s \leftarrow a(s)$
        $G \leftarrow G \cup \{s\}$
      $D \leftarrow \{(s', f_{\mathrm{t}}(s), P(s)) \mid s' \in G\}$
    `update(`f_θ, D`)`

Algorithm 2: Main algorithm of AlphaZero (see Table 1 for the definitions of symbols ; K is the number of matches performed between two updates, some of these matches are executed in parallel ; G is the sequence of states of the current match).

Note that, Descent and Unbounded Minimax are used with completion (see [Cohen-Solal, 2020] and [Cohen-Solal, 2021]), which is a minor improvement. This improvement is not described here.

2.4 Parameters of Section 3.1, 3.2, and 3.3

In this section, we present neural networks and learning parameters used in the experiments of Section "3.1 Comparison of Generated and Learning Data", Section 3.2 "Win comparison with Same Resources", and Section 3.3 "Comparison with Same Resources during a Long-Term Learning Process". These neural networks are based on convolution layer [LeCun *et al.*, 2015], residual block [He *et al.*, 2016], and ReLU [Glorot *et al.*, 2011]. We also present additional results: detailed win rate curves for each games about performance of Descent and Polygames.

Symbols	Definition
$\text{actions}\,(s)$	action set of the state s for the current player
$\text{terminal}\,(\text{s})$	true if s is an end-game state
$a(s)$	state obtained after playing the action a in the state s
$\text{time}\,()$	current time in seconds
$f_\theta(s)$	states of the partial game tree (and keys of the transposition table T)
T	transposition table (contains state labels as v or P)
$P(s)$	target policy of state s computed from the search data
D	learning data set
τ	search time per action
$t_{\max}$	chosen total duration of the learning process
$v(s)$	value of state s from the game search
$v(s,a)$	value obtained after playing action a in state s
$f_\theta(s)$	adaptive evaluation function (of non-terminal game tree leaves ; first player point of view)
$f_{\text{t}}(s)$	evaluation of terminal states, e.g. gain game (first player point of view)
action_selection(s, S, T)	decides the action to play in the state s depending on the partial game tree, i.e. on S and T
update(f_θ, D)	updates the parameter θ of f_θ in order for $f_\theta(s)$ is closer to v for each $(s,v) \in D$

Table 1: Index of symbols

Game	Reinforcement Heuristic
Surakarta	Score
Othello	Score
Breakthrough	Additive depth
Hex 13×13	Additive depth
Connect6	Multiplicative depth
Outer-Open-Gomoku	Multiplicative depth
Havannah 8	Multiplicative depth
Havannah 10	Multiplicative depth

Table 2: The games and their corresponding reinforcement heuristics (when used).

Game	F	D
Surakarta	132	845
Othello	132	477
Breakthrough	132c	477
Hex 13×13	132	155
Connect6	132	65
Outer-Open-Gomoku	132	111
Havannah 8	132	111
Havannah 10	132	65

Table 3: The filter numbers in convolutional layers and the numbers of neurons in dense layers for the R_2-networks for each game used with Descent for the 8 games experiments.

Descent Parameters

In total, four sets of parameters are used with the Descent framework: the search time per action during the learning is $1s$ or $5s$ and a reinforcement heuristic is used for half of the runs and thus a reinforcement heuristic is not used for the other half of the runs.

The reinforcement heuristic of each game is designated in Table 2 (see [Cohen-Solal, 2020] for their descriptions).

For each learning process, the batch size of the stochastic gradient descent B is 3000, *smooth experiment replay* is used with the following parameters: $\mu = 100$ and $\delta = 3$. The neural network architecture is the same for each game: a R_2-network (see Table 8). The number of weights in each architecture is of the order of $5 \cdot 10^6$. However, this implies that the number of filters F and number of dense neurons D are different for each game. The corresponding numbers are described in Table 3.

The action distribution (the action selection method) used during the training process is the ordinal law [Cohen-Solal, 2020]. It is used with a uniform random variable between 0 and 1 as exploration parameter (the variable value changes after each search performed for determining the next action to play; therefore no simulated annealing is used).

Polygames Parameters

In total, five sets of parameters are used with Polygames: three with a different neural network architecture and two others varying learning parameters.

Thus, three neural network architectures were used in this experiment for Polygames, each is an adaptation of the unique architecture being used with Descent, in order to add a policy while keeping an analogous number of weights in the neural network.

The first adds a densely connected policy after the last residual layer followed by two dense layers (dense policy network). The second adds a densely connected policy directly after the last residual layer (direct policy network). The third

adds a policy connected as a convolution layer directly after the last residual layer (convolutional policy network).

With the convolutional policy network, three sets of parameters are used:

- default: –sync_period 100 –replay_capacity 1000000 –replay_warmup 10000
- more synchronization: –sync_period 10 –replay_capacity 1000000 –replay_warmup 10000
- more synchronization and a smaller size of memory replay: –sync_period 10 –replay_capacity 100000 –replay_warmup 1000

With the two other architectures, the default set of parameters is used.

In Table 4, the number of filters in convolutional intermediate layers and neurons in dense intermediate layers for each used neural network architecture and for each game are described.

For the evaluations versus the basic MCTS based on UCT with 160 rollouts, 400 games were performed for each learning process (i.e. each learned neural network). Thus, the 95%-confidence radius is max 5%.

Performance variation due to variation in Polygames architectures and settings is small and game-dependent.

Detailed Results

The performances against a 160-rollouts MCTS (with UCT and without any knowledge nor learned policy) of the Descent framework and respectively of Polygames are described in Table 5. The associated average win percentage and the stratified bootstrap confidence interval over all games are described in Table 6.

Win rate curves of Descent and Polygames against MCTS with UCT, corresponding to the experiment 3.2 are detailed for each game in Figure 1 and Figure 2. For the curve of the paper and these additional curves, there is an evaluation each day of the 15 days and an evaluation halfway through the first day (at 12 noon). An evaluation consists of 200 matches against MCTS as the first player and another 200 matches against MCTS as the second player.

2.5 Detailed Learning Data Analysis

In the main paper, we have analyzed the number of learned states. Here, we are interested in what we call *learned numbers*, which is an alternative measure. Then, we give a detailed conclusion, notably comparing the two measurements.

Thus, we compare *learned numbers*, which are the numbers evaluating the states. For Descent, the number of learned numbers is the number of learned states (i.e. the number of their values), but this is not the case for Polygames where a value and a policy are learned for each learned state. Recall that a policy is a set of probabilities, one for each action in that state. Thus, for Polygames, the number of learned numbers is the number of learned probabilities plus the number of learned values (i.e. the sum of the sizes of the learned policies plus the number of learned states).

In other words, contrary to the number of learned states which is the number of states s such that the adaptative evaluation f_θ has been updated to fit $y = f_\theta(s)$, the number of

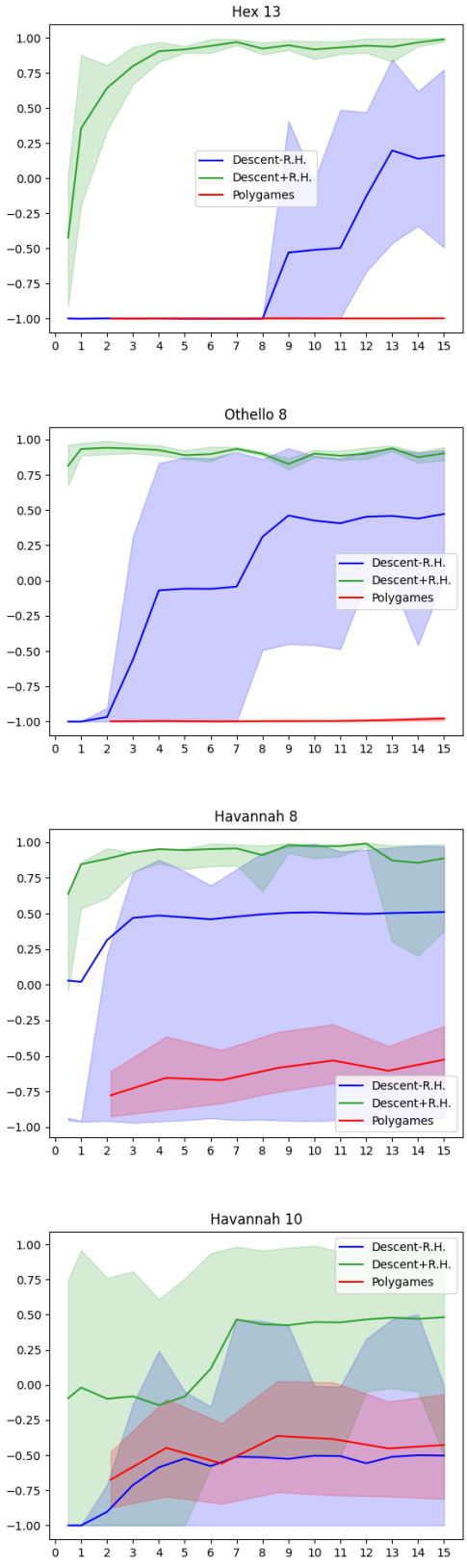

Figure 1: Evolution of win percentage of Descent with reinforcement heuristic (+R.H), Descent without reinforcement heuristic (-R.H), and Polygames against MCTS (with UCT) along the 15 days of training and their confidence intervals for the first 4 games.

policy network	dense		direct		convolutional		
layer type	conv.	dense	conv.	dense	conv.	dense	c
Surakarta	$83c$	83	$42c$	42	$88c$	88	2
Othello	$78c$	78	$74c$	74	$85c$	85	2
Breakthrough	$78c$	78	$69c$	69	$85c$	85	2
Hex 13×13	$60c$	74	$46c$	74	$74c$	74	2
Connect6	$10c$	28	$2c$	160	$11c$	42	13
Outer-Open-Gomoku	$38c$	62	$24c$	46	$50c$	52	3
Havannah 8	$43c$	46	$25c$	25	$50c$	52	3
Havannah 10	$38c$	38	$12c$	12	$46c$	46	3

Table 4: Number of filters in each intermediate convolutional layers and number of neurons in intermediate dense layers for each used neural network architecture and for each game used with Polygames in the 8 games experiments ; c is a Polygames game constant.

Mean		Connect6	Havannah 10	Havannah 8	Gomoku	Hex 13	Surakarta	Othello	Breakthrough
D	win	0.0 ± 0.0	24.8 ± 48.7	51.0 ± 54.1	100 ± 0.0	58.1 ± 42.3	67 ± 10.0	73.4 ± 42.6	100 ± 0.0
	loss	100 ± 0.0	75.1 ± 48.7	49.0 ± 54.1	0.0 ± 0.0	41.8 ± 42.3	0 ± 0.0	26.1 ± 42.2	0.0 ± 0.0
D_{RH}	win	60.5 ± 43.2	74.0 ± 48.4	88.6 ± 19.7	99.2 ± 0.5	99.5 ± 0.8	98 ± 0.9	95.1 ± 3.0	75 ± 48.9
	loss	39.4 ± 43.2	25.9 ± 48.4	11.3 ± 19.7	0.7 ± 0.5	0.4 ± 0.8	0 ± 0.0	4.8 ± 3.1	25 ± 48.9
P	win	1.3 ± 1.9	28.6 ± 23	23.7 ± 12.6	38.1 ± 13.8	0.1 ± 0.1	91.2 ± 4.7	0.9 ± 0.8	18.5 ± 13.3
	loss	98.7 ± 1.9	71.3 ± 23	76.3 ± 12.6	61.8 ± 13.8	99.8 ± 0.1	5.1 ± 2.4	98.8 ± 0.9	81.4 ± 13.3
Gain of D		-1.3%	-3.8%	27.3%	61.85%	58%	-9.55%	72.6%	81.45%
Gain of D_{RH}		59.25%	$45,4\%$	64.95%	61.1%	99.4%	5.95%	94.1%	56.4%

Table 5: Results in percentage of UBFM_s (resp. Polygames) using the learned neural nets from the Descent framework (resp. Polygames) with 15 days of training against MCTS with UCT (no provided nor learned knowledge for the MCTS) and gain of using Descent rather than Polygames (see Def. **??**). The 95%-confidence radius is indicated by $\pm$.

	Polygames		Descent			
			without R.H.		with R.H.	
	mean	S.B.C.I	mean	S.B.C.I	mean	S.B.C.I
win	25.3%	$[22.2\%; 28.3\%]$	59.2%	$[50.5\%; 68\%]$	86.2%	$[78.3\%; 93.6\%]$
loss	74.1%	$[71.2\%; 77.2\%]$	36.4%	$[27.8\%; 45\%]$	13.5%	$[6.2\%; 21.5\%]$

Table 6: Average results over the 8 games and the associated stratified bootstrap confidence interval (abbreviated S.B.C.I) of UBFM_s (resp. Polygames) using the learned neural nets from the Descent framework (resp. Polygames) during 15 days against MCTS with UCT (no provided nor learned knowledge for the MCTS). Descent results are detailed in function of the use of a reinforcement heuristic (abbreviated R.H.)

learned numbers is the number of learning states times the dimension of outputs y (i.e. for Polygames the number of learned probabilities of actions plus the number of learned states ; for Descent it is just the number of learned states).

During the experiment in the article measuring the learned states, we also measured the learned numbers. The conclusion for learned numbers is not the same as for learned states. The value ratio for each game has been notified in the Table 7 (which also recalls the data results of the experiment of the main article). Descent is from 63 times better to 5 times worse depending on the game. This is obviously on games with a large branching factor that Descent has less good results. In Havannah 8 and Outer-Gomoku, the results are slightly lower (they should however be at least slightly higher by changing the programming language). On the other hand, it is possible that on Connect6 and Havannah 10, even by changing the programming language, the numbers are better with Polygames. Parallelizing Descent might fix this again. Even so, a probability of playing the action leading to a child state is not equal to a state value. In summary, on the majority of games, Descent is more data efficient than Polygames. But on games with high branching factor Polygames is more efficient only compared to learned numbers.

In conclusion, the cost of generating the learned numbers is only significantly higher for 4 of the 8 games, and significantly lower for 3 of the 8 games. Nevertheless, the question arises as to whether the numbers of Polygames contain the same level of information as the numbers of Descent.

Note that, there is a correlation in the game tree data. Therefore, the measures of learning data cannot be used to conclude on the superiority of one algorithm over another. The study of these values shows us that there is a real leap in terms of the number of learned data (in particular about learned states) between the two methods, which only reinforces the difficulty of comparing the two methods, other than by studying the win percentage. However, despite the redundancy, the level of gained information, although not quantifiable, is most likely very significant, given the huge difference

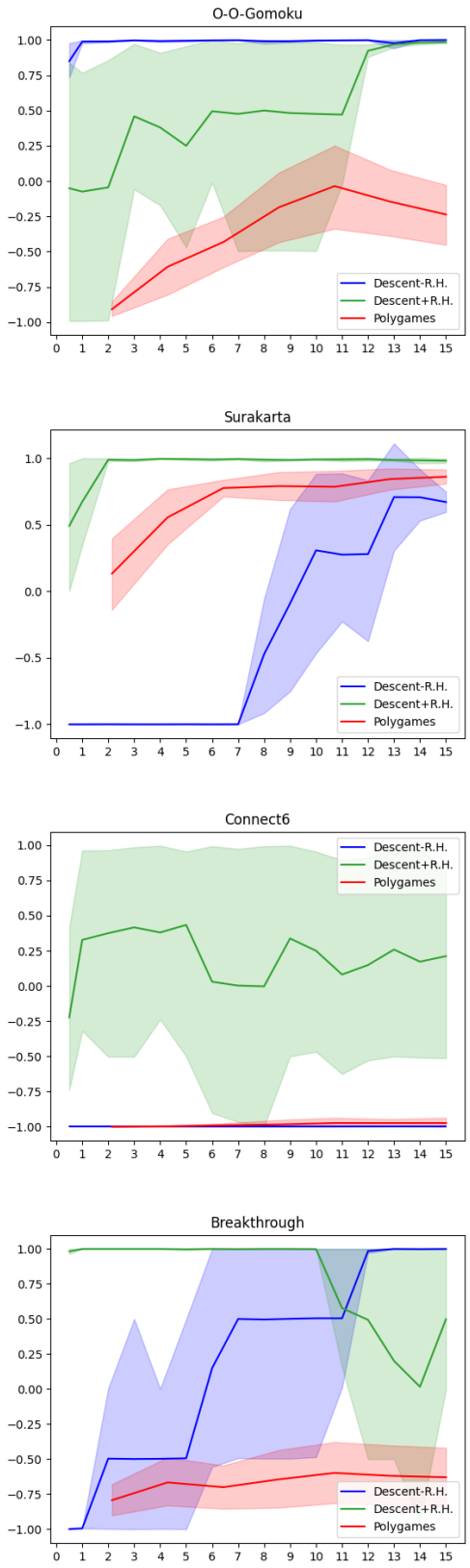

Figure 2: Evolution of win percentage of Descent with reinforcement heuristic (+R.H), Descent without reinforcement heuristic (-R.H), and Polygames against MCTS (with UCT) along the 15 days of training and their confidence intervals for the last 4 games.

in generated data between the two methods.

2.6 Common Neural Networks and Parameters for the Second Longer Run and the Tournament Polygames Experiment

We present the common neural networks and parameters used in the next sections.

We use two sets of learning parameters for the Descent framework in the experiments of the main article and those of document. The set of parameters, that we call *A-parameters*, is: search time per action $\tau = 1s$, batch size $B = 3000$, the number of states in the learning memory size is $2 \cdot 10^6$, sampling rate $\sigma = 5\%$ (for more details on these parameters, see Section 3 of [Cohen-Solal, 2020]). The set of parameters, that we call *B-parameters*, is $\tau = 2s$, $B = 3000$, the states in the learning memory are those of the 250 last matches, and $\sigma = 2\%$. Moreover, B-parameters includes the use of the *data augmentation from the symmetry of the board* and the *color board sides encoding* (see Section 7.2.1 of [Cohen-Solal, 2020]).

When the depth heuristic is used with Hex 11×11, the corresponding neural networks are in $[-121; 121]$ (by using the depth heuristic, terminal states are evaluated based on the duration of games, which lasts, in the case of Hex 11, at most 121 turns).

Moreover, we respectively use one neural network from the Descent framework, training during 30 days, for each of the following games: Breakthrough, Othello with board size 8×8, and Othello with board size 10×10. The learning parameters are the A-parameters. At Othello 8 and 10 the scoring heuristic is used, at Breakthrough the depth heuristic is used. The values of the Breakthrough (resp. Othello 8 ; resp. Othello 10) neural network is in $[-481; 481]$ (resp. $[-64; 64]$; resp. $[-100; 100]$). All Descent networks are pre-initialized by the values of random terminal states (around 10^7, i.e. a supervised learning is performed for terminal states from random games. The neural network architectures used in this article are described in Table 9.

2.7 Another Comparison with Same Resources during a Long-Term Learning Process

In this section, we compare again the learning performances of the Descent framework with the learning performances of Polygames with respect to the win percentages, on a rather long training (30 days), only at Hex 11, and against Mohex 2.0 [Huang *et al.*, 2013], champion program at Hex from 2013 to 2017 at the Computer Olympiads, the strongest hex program that is freely available.

Several training runs have been carried out on the game Hex (size 11). Two training using Descent have been performed with respectively the depth heuristic and the classic gain of a game (-1 / $+1$) as terminal evaluation (i.e. with reinforcement heuristic and without reinforcement heuristic). Unlike these two Descent training runs, the Polygames training runs does not use the *data augmentation from the symmetry of the board* and the *color board sides encoding* (see Section 7.2.1 of [Cohen-Solal, 2020] ; note that these two techniques have not been used in the previous sections). For this reason, a third learning run using the Descent framework was

	Connect6	Havannah 10	Havannah 8	Gomoku	Hex 13	Surakarta	Othello	Breakthrough
Learned states	55	64	111	115	359	442	529	693
Neural evaluation	0,02	0,03	0,05	0,04	0,10	0,11	0,12	0,16
States evaluation	0,37	0,30	0,37	0,49	0,65	0,40	0,10	0,49
Learned numbers	0,17	0,26	0,78	0,58	2,8	20	63	27

Table 7: Ratio of Descent data over Polygames data for the same learning time for different games and 5 runs for Polygames et 4 runs for Descent (data of a run varies by a maximum of $\pm 60\%$ for Polygames and $\pm 20\%$ for Descent). For example, in Connect6, Descent learns 55 times more states, makes 50 times less neural evaluations, makes 3 times less state evaluations, and learns around 5 times less numbers.

layer #	C-network	R_1-network	R_2-network
1	conv. + ReLU	convolution	convolution
$\cdots$	conv. + ReLU	2 res. blocks	8 res. blocks
$N-2$	conv. + ReLU	1×1 conv.	dense + ReLU
$N-1$	dense + ReLU	dense + ReLU	dense + ReLU
N	dense layer	dense layer	dense layer

Table 8: Description of 3 neural architectures of value networks, called C-network, R_1-network, and R_2-network. Each residual block is composed of a ReLU followed by a convolution followed by a ReLU followed by a convolution followed by a ReLU. Output contains one neuron. Other parameters are: kernel is 3×3, filter number is F, neuron number in dense layers is D, with padding for R_i-network and without padding for C-network.

network	architecture	F	D
Breakthrough	C-network	166	751
Othello 8	R_2-network	59	213
Othello 10	R_2-network	59	128

Table 9: Description of the used value neural network architectures in the experiments about Polygames tournament networks of the paper (F and D are parameters of the architecture ; see Table 8).

performed without using these two techniques. The depth heuristic is also not used. Another analogous learning run with Descent has also been performed using only the symmetry augmentation (i.e. sides encoding and depth heuristic are not used). Two training runs have been performed with Polygames, each using a different neural architecture.

The evolution of the win percentages against Mohex 2.0 for each program during the learning process is shown in Figure 3. For this experiment, learning with *Descent* is widely better than with Polygames. The performance gain is more marked at the start of learning: Descent reaches a high level from the beginning of the training run.

Technical Details

The Descent learning processes have used 2 CPU per run and one GPU for all runs (4 training runs were launched in parallel on the same GPU for Descent, resulting in a performance loss between 10% and 20%). For Polygames, 10 CPU and one GPU were used per run. The first Polygames network uses a similar network architectures than those for Descent, adapted to provide a policy while having the same number of weights. The second Polygames network uses an AlphaZero-like architecture with a similar number of weights. The search time for each program is 2 seconds. The search algorithm used to evaluate the Descent networks is UBFM_s. The Polygames program is used to make the trained Polygames networks play (i.e. MCTS with PUCT). Note that, in the context of these experiments, the Polygames program has performed 3 times more state evaluations than the Descent program (thanks to its parallelization of games and its C/C++ implementation).

The Descent framework was used to train a first neural network, by using the B-parameters, the depth heuristic, and the C-network architecture.

The first Polygames network has 40 filters, 50 dense neurons, and its policy is densely connected to the last intermediate layer.

The second Polygames network (that with the AlphaZero architecture) has 8 residual blocks with 64 filters and 256 neurons in the dense layers.

The parameters of Polygames training runs are the default parameters except that num_game=121 and act_batchsize=121 (they are increased to improve the parallelism ; num_game is the number of games performed in parallel and act_batchsize is the number of evaluations batched together).

The third and the four learning processes based on the Descent framework use the A-parameters.

2.8 Polygames Tournament Neural Networks

In the experiments of Section 3.4 of the article, we use particular Polygames neural networks which have participated in the TCGA tournament of 2020 (in Taiwan): the Breakthrough network won the tournament, the Othello size 8 network finished second, and the Othello size 10 network won the tournament. Their training runs required the use of 100 to 300 GPU and 80 CPU during a week.

Parameters for Breakthrough and Othello

In this subsection, we describe the details of the Descent training for Othello (size 8 and 10) and Breakthrough.

The training run using the Descent framework on Othello 8 and Breakthrough is the one described in Section 2.6 (the first 6 days of the 30 days of training). Note: neither symmetry nor sides coding is used.

Detailed Results

Finally, we present details about the results of Section 3.4.

The results of the confrontation of the 5-day Descent networks against Polygames networks are described in Table 10 (This table presents the results of 200 matches as first player and 200 other matches as second player of the Descent networks of Section 2.6 with 5 days of learning, using UBFM_s

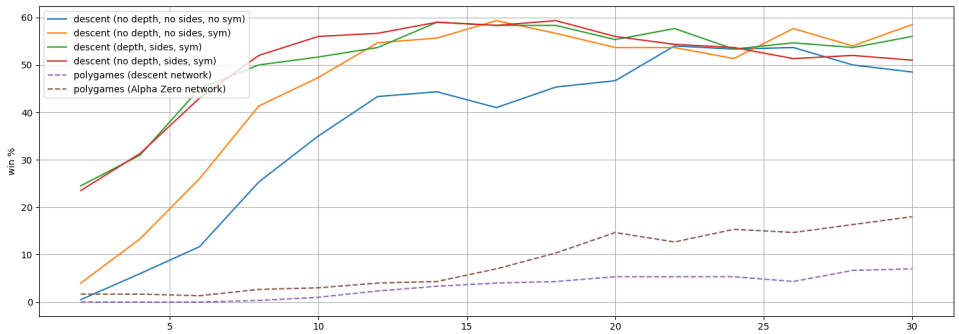

Figure 3: Evolution of winning percentages of Descent with classic gain, symmetry augmentation and sides encoding (red line), Descent with depth heuristic, symmetry augmentation and sides encoding (green line), Descent with classic gain and symmetry augmentation (orange line), Descent with classic gain (blue line), Polygames using the corresponding Descent network architecture (purple dotted line), and Polygames using an AlphaZero neural architecture (brown dotted line) against Mohex 2.0, during a 30 days learning process (approximately one evaluation every two days ; each evaluation consists of 300 matches in first player and 300 other matches in second player). The x axis is in days.

	time	Breakthrough		Othello 8		Othello 10	
days		5	30	5	30	5	30
win	$1.5s$	56%	85%	65%	97%	67%	98%
draw	$1.5s$			1%	2%	1%	1%
loss	$1.5s$	44%	15%	31%	1%	32%	1%
win	$5s$	58%	55%	81%	94%	61%	79%
draw	$5s$			3%	4%	3%	2%
loss	$5s$	42%	45%	16%	2%	36%	19%

Table 10: Results of 400 matches of Descent networks (5 days of learning and 30 days of learning) with UBFM_s at Breakthrough and Othello 8 and 10 against tournaments Polygames networks (see Technical Appendix). Confidence radius: max 5%.

as search algorithm at Breakthrough and Othello (size 8 and 10) against tournaments Polygames networks). Although learning with Descent used 100 times less GPU (1 GPU vs. 100 GPU) and lasted slightly less time (5 days vs. a week), the Descent framework has a much better result for all three games.

The results of the confrontation of the 30-day Descent networks against the same Polygames networks are described in Table 10 (This table presents the results of 200 matches as first player and 200 other matches as second player of the Descent networks of Section 2.6 with 30 days of learning, using UBFM_s as search algorithm at Breakthrough and Othello (size 8 and 10) against tournaments Polygames networks). This new experience shows that the Descent networks continue to improve.

2.9 Questions of Reviewers

In this section, we provide answers to questions from reviewers of previous submissions.

- How would one quantify the complexity of the MCTS-based algorithm and the Minimax-based method, for instance?

MCTS algorithms and Minimax algorithms are heuristic algorithms that usually do not give any guarantee. When used in the endgame they can solve the subgame but this is not the general case. Even for solving the subgame the worst case complexity is that they explore the whole subtree since they are not guaranteed to have a good move ordering. Qualifying the complexity in the case of MCTS and Minimax is ill-defined since the result of running the algorithms is a move that is supposed to be good but not a perfect answer to a well defined problem. In order to compare MCTS and Minimax the most meaningful comparison is to make them play against the same adversary and see the difference in winning rate. A possible measure of complexity is the number of states visited for a given time. Both MCTS and Unbounded Minimax perform an iteration to expand the most interesting leaf. However in the case of Unbounded Minimax all the children of the most promising leaf are expanded while in the case of MCTS only one child is expanded. In the worst case in order to solve a game MCTS would visit much more states than Unbounded Minimax since MCTS expands one state per iteration whereas Unbounded Minimax expands all the possible moves of the leaf per iteration. Therefore the number of iterations in the worst case to solve a game in the case of Unbounded Minimax is much less than the number of iterations of MCTS in the worst case to solve a game.

Note also that a complete analysis that integrates learning in addition to search is extremely complicated, especially since the two algorithms have incomparable characteristics (different learning target values, use / non-use of policy, .. .).

- How applicable is the Decent algorithm? What are its limitations? How can you improve it?

Descent can be applied to any two-player game with perfect information (and no chance). In this case, with enough time and memory, it finds a winning strategy [1]. It can be applied to games with imperfect information (in particular stochastic games) and to multi-player games but it will not have, in general, an optimal behavior. Thus, we can improve it so that it has a theoretically optimal behavior with multiplayer games and imperfect information games.

Its practical limitations are an open question. It gives good to excellent results on more than fifteen games. It's only in

Chinese checkers where the results are disappointing (at least for now), which is surprising since the results are very good on classic Checkers.

References

[Bonnet *et al.*, 2013] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. Havannah and twixt are pspace-complete. In *International Conference on Computers and Games*, pages 175–186. Springer, 2013.

[Bonnet *et al.*, 2016] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. On the complexity of connection games. *Theoretical Computer Science*, 644:2–28, 2016.

[Cohen-Solal, 2020] Quentin Cohen-Solal. Learning to play two-player perfect-information games without knowledge. *arXiv preprint arXiv:2008.01188*, 2020.

[Cohen-Solal, 2021] Quentin Cohen-Solal. Completeness of unbounded best-first game algorithms. *arXiv preprint arXiv:2109.09468*, 2021.

[Glorot *et al.*, 2011] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *The Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[Hsieh and Tsai, 2007] Ming Yu Hsieh and Shi-Chun Tsai. On the fairness and complexity of generalized k-in-a-row games. *Theoretical Computer Science*, 385(1-3):88–100, 2007.

[Huang *et al.*, 2013] Shih-Chieh Huang, Broderick Arneson, Ryan B Hayward, Martin Müller, and Jakub Pawlewicz. Mohex 2.0: a pattern-based mcts hex player. In *International Conference on Computers and Games*, pages 60–71. Springer, 2013.

[Iwata and Kasai, 1994] Shigeki Iwata and Takumi Kasai. The othello game on an n$\times$ n board is pspace-complete. *Theoretical Computer Science*, 123(2):329–340, 1994.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Piette *et al.*, 2020] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – the ludemic general game system. In G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 411–418. IOS Press, 2020.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.