Implementation and Evaluation of Information Set Monte Carlo Tree Search for *Pokémon*

Hiroyuki Ihara*, Shunsuke Imai*, Satoshi Oyama*†, and Masahito Kurihara*
*Graduate School of Information Science and Technology, Hokkaido University
†Global Institution for Collaborative Research and Education, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan
{ihara_h, imai_s}@complex.ist.hokudai.ac.jp, {oyama, kurihara}@ist.hokudai.ac.jp

***Abstract*—Artificial intelligence has shown remarkable performance in perfect information games. However, it is still no match for human players when it comes to most imperfect information games. Information set Monte Carlo tree search (ISMCTS) has been developed to reduce the effects of strategy fusion caused by determinization of the imperfect information and demonstrated advantages over the conventional Monte Carlo tree search (MCTS) that uses determinization. Because ISMCTS has only been used for games with relatively simple structure, it is still unknown whether it works effectively for more complex games. In this study, we take *Pokémon* as an example of a complex imperfect information game and implement a simulator to evaluate the effectiveness of ISMCTS. Experimental results show that ISMCTS outperforms the conventional MCTS that uses determinization.**

***Index Terms*—Imperfect information games, game AI, strategy fusion, Information Set Monte Carlo Tree Search (ISMCTS), Pokémon**

I. Introduction

Computer programs for games have existed since the early years of artificial intelligence (AI) research. Owing to the advancement of computer hardware, AI, and machine learning algorithms, todays computers can play various kinds of games, from easy games, such as Rock-paper-scissors and Tic-tac-toe, to complex classic board games, such as chess, Reversi, Japanese chess, and Go, as well as video games. Since playing smart games is considered a sign of intelligence, game playing has been used as a benchmark for the AI systems and has attracted considerable attention from many researchers.

Games are divided into classes based on their features. Among them, perfect information games are the class of games where the players are completely informed of all the states of the game. After long years of research, the game AI of most perfect information games is as strong as human professionals. Machine learning algorithms to learn from the enormous data of human playing records and efficient search algorithms such as Monte Carlo tree search (MCTS) were very important technical advancements. For example, AI for the game of Go [1] defeated a world champion using a method that made full use of Monte Carlo tree search and deep learning in 2016. And in the next year, another AI [2] defeated the previous one after training for only a day and a half with a machine learning method that does not use the human playing records.

However, in most of the imperfect information games where players cannot completely grasp the states of the game, AI still cannot compete with human professionals because it cannot learn and search the missing information. Since the ability to deal with missing information is one of the notable characteristics of human intelligence, building AI that can play imperfect information games is one of the important challenges for artificial intelligence research.

For perfect information games, Monte Carlo tree search has been applied with great success. MCTS is a tree search algorithm that incorporates the characteristics of a tree search in the classical Monte Carlo method. MCTS is different from the Monte Carlo method in two major points. First, MCTS uses a different method to choose regal moves in random playouts. The classical Monte Carlo method allots playouts equally to all regal moves from the root node, while MCTS generates more playouts for regal moves with higher winning rates. Second, in MCTS each node remembers the number of playouts performed for itself, and new child nodes are generated when the number of playouts exceeds a threshold. Therefore, the game tree of MCTS grows as the number of playouts increases.

Since MCTS is an algorithm designed for perfect information games, it cannot be used for imperfect information games as it is. One can use MCTS for imperfect information games with the *determinization* technique, which presumes the values of unknown information as if they were known, in order to transform an imperfect information game into a perfect information game. The determinization technique has been successfully used for games such as *Bridge* [3] and *Klondike Solitaire* [4]. However, we know that MCTS in imperfect information games is weak when it is strongly influenced by *strategy fusion* [5]. Information set Monte Carlo tree search (ISMCTS) [6] has been proposed to solve this problem. ISMCTS replaces the nodes in MCTS with *information sets* to avoid the problem of strategy fusion.

However, ISMCTS has only been used for games with relatively simple structure so far, and it is still unknown whether it works effectively for complex games which have more states and branches. Therefore, we took *Pokémon* as an example of a complex imperfect information game and evaluated the effectiveness of ISMCTS for it. Specifically, we have developed a simulation platform for *Pokémon* and conducted experiments to compare ISMCTS with several baseline algorithms.

II. Background

As mentioned in Section I, MCTS can allot more playouts to more promising moves. In addition, MCTS avoids moves that do not yield a high winning rate even if they initially looked promising because of the mistake made by the opponent. However, MCTS has to deal with the *multi-armed bandit problem*, since it chooses the moves considered to be promising when it generates the playouts. Multi-armed bandit is a problem which has been studies in various fields such as statistics. When one is given several slot machines for which the expected reward is unknown, one has to develop a policy to choose the machines to bet money on in order to maximize the profit under budget constraints. In the case of MCTS, one has to consider how to allot playouts to the moves for which the true winning rates are unknown. Multi-armed bandit problem has two large factors to consider: *exploration* and *exploitation*. Exploration is choosing a move that has not been tried many times before to obtain the information on its expected reward. Exploitation is choosing the most promising move based on the available information at a particular time point to secure the profit. Since exploration and exploitation are in a trade-off relationship, it is important to find a policy that can keep the balance between them.

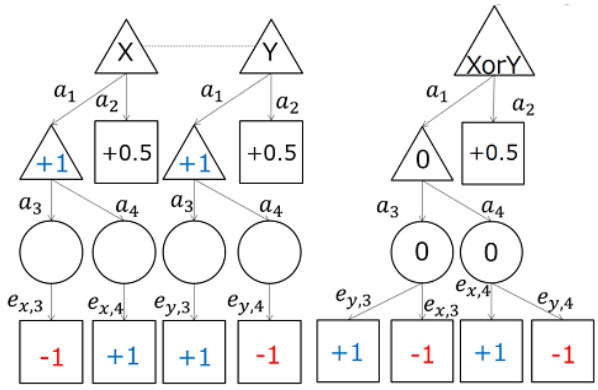
Auer *et al.* have developed a well-known algorithm called upper confidence bound (UCB) [7] that can balance the trade-off between exploration and exploitation. UCB chooses the machine which has the highest value of $x_j + \sqrt{(2\log n)/n_j}$ as the next investment, where x_j is the winning rate of the slot machine j, n_j is the number of bets for the slot machine j, and n is the total number of bets. The first term represents exploitation, the second term represents exploration, and both are considered under a moderate balance. UCB converges to the true winning rate as the number of bets grows, since the second term becomes approximately 0 if n becomes large enough.

UCB applied to trees [8] (UCT) is an algorithm that utilizes UCB to choose the next node and the mainstream in MCTS. UCT uses UCB as $x_j + c\sqrt{(2\log n)/n_j}$ where c is the exploration constant which has to be adjusted for each game. After the introduction of the UCT algorithm, AI for perfect information games with large game trees became much stronger, as can be seen, for example, in the success of AI for the game of Go.

III. Information Set Monte Carlo Tree Search

UCT is effective for perfect information games but not for some imperfect information games which suffer from the effects of strategy fusion. The effects of strategy fusion means choosing wrong strategies when the values of imperfect information are presumed to be observed, and the game tree is solved deterministically. We explain the effects of strategy fusion by using a simple game (based on Cowling *et al.* [6]).

This game is played by a single player; the environment, either X or Y, is determined with a probability of 50% at the beginning of the game, and the player cannot know the value of the environment. The player can choose either move a_1 or move a_2 at the beginning of the game. The game finishes immediately if he chooses a_2 and the player receives a payoff of $+0.5$. On the other hand, if he chooses a_1, he can then choose a_3 or a_4. If he chooses a_3, he receives -1 if the environment is X, or he receives $+1$ if the environment is Y. If he chooses a_4, he receives -1 if the environment is Y, or he receives $+1$ if the environment is X. Fig. 1 (a) shows a game tree that uses determinization, so it is affected by strategy fusion. If the player presumes that the environment is X and chooses a_1, he will always chose a_4 if he is rational, since he will receive -1 if he chooses a_3 and $+1$ if chooses a_4. A rational player always chooses a_3 and receives $+1$ for the similar reason if he presumes the environment is Y. Thus, with the use of determination, the expected payoff of the player choosing a_1 becomes $+1$, which is larger than the $+0.5$ payoff when choosing a_2.



(a) With strategy fusion (b) Without strategy fusion

Fig. 1. Game trees for a simple game (based on Cowling *et al.* [6])

On the other hand, Fig. 1 (b) shows a game tree that takes into account the fact that the player cannot know whether the environment is X or Y, which strictly follows the original game rule. Since the payoff when the player chooses a_1 and a_3 is $+1$ with a probability of 50% and -1 with a probability of 50%, the expected payoff is 0. The payoff when the player chooses a_1 and a_4 is 0 for the similar reason. Thus, using this game tree, the expected payoff when the player chooses a_1 is 0. A rational player should choose a_2 since the payoff for a_2 is bigger than the payoff for a_1. However, under determinization, the player considers which move to choose according to the game tree in Fig. 1 (a) and he will choose a wrong move. This is the effect of strategy fusion and it occurs frequently when determinization is used for imperfect information games.

One of the methods to reduce the effects of strategy fusion is changing each node representing a state in a game tree to represent an information set. An information set is a collection of states which a player cannot distinguish. Information set Monte Carlo tree search (ISMCTS) enables the game tree search that reduces the effects of the strategy fusion by adopting the information sets in MCTS. Cowling *et al.* proposed three versions of ISMCTS [6].

The first version is a single-observer information set MCTS

(SO-ISMCTS). This is the most basic form of ISMCTS with all nodes as information sets from the viewpoint of the root node player. The effect of strategy fusion cannot be completely avoided by SO-ISMCTS because it treats unobservable actions of the opponent player as observable.

The second version is a single-observer information set MCTS with partially observable moves (SO-ISMCTS+POM), which assumes that the opponent player chooses his actions randomly. Because SO-ISMCTS+POM searches the game tree without assuming partially observable moves as completely observable, it is less affected by strategy fusion in the case of games with partially observable moves. However, the modeling of the opponent player in SO-ISMCTS+POM is poor because it treats all partially observable moves equally.

The third version is a multiple-observer information set MCTS (MO-ISMCTS). This version maintains a separate tree for each player; the nodes of each tree correspond to the players information sets; the search for all trees is performed simultaneously. MO-ISMCTS has a better modeling of the opponent player than SO-ISMCTS+POM. Since the second and third versions treat the imperfect information more carefully, they are expected to show better performance in all games. But Cowling *et al.* demonstrated in their experiments that it was not always the case [6].

Cowling *et al.* adopted *exponential-weight algorithm for exploration and exploitation* (EXP3) [9] instead of UCB in the simultaneous move nodes in the performance evaluation experiment of ISMCTS. An optimal policy of the simultaneous move node is often a mixed strategy, and EXP3 finds such mixed strategy, while UCB is designed to converge in a pure strategy. EXP3 uses the roulette wheel selection to select the arm a based on the probability:

$$p(a) = \frac{\gamma}{|A|} + \frac{1-\gamma}{\sum_{b \in A} e^{(s(b)-s(a))\eta}},$$

where $\gamma = \min(1, \sqrt{|A| \log |A| / ((e-1)n)})$, $\eta = \gamma / K$, $|A|$ is the number of regal moves in the node, and $s(a)$ is the sum of rewards for selecting arm a.

In the information set for which the acting player does not equal to the observer of it, different states in the set can have different sets of legal moves. For example, different states in an information set for which the opponent player acts may have different sets of legal moves since possible opponent moves in a card game depend on the hand of the opponent player. Thus, an arm of such information set nodes is a logical sum of the sets of legal moves for all states in the information set, but the sets of legal arms at each iteration are different. In other words, it is like a subset of the multi-armed bandit. Thus, Cowling *et al.* called this the *subset-armed bandit* [6]. If the acting player of the information set does not equal to the observer of it and the information set has arms which are legal and of low frequency, whenever such arms become legal, their values of UCB become large and they are selected unfairly. If every state in the information set has a rare arm, ISMCTS focuses mostly on exploration and does not consider exploitation. In order to avoid such a situation, Cowling *et al.* changed n in UCB from the total number of trials to the number of times the node was available.

IV. Characterization of Pokémon

Pokémon[1] is a *turn-based game* which is played by two players and consists of simultaneous move nodes. A turn-based game is divided into explicit parts called *turns*. Players use monsters called *Pokémon* as hand pieces. Each player has a field monster and two reserve monsters. At each turn in the game, the player chooses a single move from his own field monsters' moves and uses it against the opponent monster, or exchanges his own field monster with one of his own reserve monsters. If the move successfully hits the opponent monster, the opponent monster receives damage and the value (called HP) of it decreases. When HP of a monster becomes zero, it faints and cannot act. At the time of the turn end, each player replaces his own fainting monster with one of his reserve monsters that did not faint. The goal of a player is to make all opponent monsters faint before all of his own monsters faint.

The set of rules for *Pokémon* which we deal with in this paper is called *the single flat rule of sixth generation*. We describe the main rules below.

- Players can use all monsters except legendary monsters and mythical monsters.
- Players prepare six different monsters each before the game starts.
- A monster whose *level* is from 51 to 100 becomes a *level* 50 monster.
- Players cannot let two or more monsters have the same *item*.
- Each player discloses all of his monsters to the opponent player before letting the monsters fight after the game starts.
- After the player sees the opponent monsters, he has to choose three monsters including the first field monster within two minutes.
- A single monster acts in each player's battle field.
- At every turn, each player has one minute to consider the next move.
- After the game time of 30 minutes is up, the game ends, and the winner and the loser are decided by special rules.

Pokémon is a zero sum game with imperfect and uncertain information which is played by two players. Imperfect information is more important in this game than in *Mahjong* and poker which were studied for a long time. One does not become weak in this game even if one does not think about the concrete opponents hand. However, it is essential to assume the parameters of the opponent monsters concretely to evaluate the moves in *Pokémon*, since the value of damage of a move against the opponent monster uses the parameters of the opponent monster. If the parameters are always constant, there is not a large problem. However, the players can set them freely to some extent, and these parameters are the hidden

[1] *Pokémon* is a trademark of Nintendo.

information at the beginning of the game. Thus, we need to presume the information of the opponent monsters concretely.

In addition, this game has the uncertainty of information. Poker, *Japanese playing cards*, and Mahjong are also games with uncertain information. However, the uncertainty of information in these games is eliminated in most cases by shuffling the deck at the beginning of the game. When we search for a game tree, most of the uncertainty disappears when we presume the imperfect information concretely at the time of the start of the search. The branches rarely depend on luck during the rest of the search. However, *Pokémon* depends on luck in almost every scene. We can say that most actions are affected by luck in *Pokémon*: whether a move hits the opponent successfully, how much damage the opponent gets, whether an additional effect of a move becomes effective, whether an item or an ability become effective, all of that depends on luck. The uncertainty of information greatly influences the search for the game tree and makes the game tree large. Another problem is that it slows the convergence of the winning rate.

One has to presume the values for various kinds of parameters when playing *Pokémon*, and the ranges of the values of these parameters are large. However, possible values of the parameters that a good player will choose are limited. It might be possible to reproduce such expert's knowledge, but it is not the main subject of this study. We used another method to assume the parameters. We obtained the statistics showing how good players choose the parameters from an official web site called *Pokemon Global Link* (PGL)[2] and used it in presuming the parameter values of the opponent. The PGL data has records of the top 10 probabilities of moves for each species of monsters, their abilities, *natures*, and items. When the algorithm searches for a game tree from the root node, it presumes the parameters concretely using these data. If the parameters for a monster do not exist in PGL, we presume their values based on our knowledge about the game.

V. Implementation

To evaluate the algorithms, we developed a simulator and reproduced the environment that followed the single flat rule of sixth generation. Generally, a program to search a game tree needs two functions: a function to make the next state of the game according to the game rule and a function to memorize the current state of the game. In the case of a perfect information game, the viewpoint of all players accords with a viewpoint of God, and all players hold perfect information about the state of the game. Since the node of the search tree is the node from the viewpoint of a player, it contains all the information about the state of the game at a particular time. Therefore, the class which memorizes the node of the game can expand the nodes. In other words, in the implementation of a search tree in the perfect information game it is not beneficial to divide the class that memorizes the state of the game and the class that is responsible for making the next state of the game according to the game rule.

[2]https://3ds.pokemon-gl.com/

On the other hand, the program to search a tree of an imperfect information game with uncertainty does not need to let nodes hold the perfect information if one assumes that the node represents an information set. In that case, one can save memory space and reduce the calculation cost when one copies a node class. This is because the node class that memorizes the state of the game holds only the information to constitute the information set. However, a simulator class which expands the next nodes from the rule of the game and the current state of the game is necessary since the node class of the tree search does not hold the perfect information about the state of the game. In this way, we can divide a simulator class, which makes the next state of the game, and a node class, which memorizes the state of the game, when we search a tree of the imperfect information game with uncertainty.

In *Pokémon*, one can rarely make a node using only the information that an information set holds. In addition, pruning is necessary in every playout since the moves of the opponent in the opponent player node depend on the imperfect information as in the card games and Mahjong. For these reasons, communication between the node class and the simulator class in the simulator of *Pokémon* becomes very frequent if one divides them. If the active node has the perfect information of the game, one can search the game tree with only the node class and can copy the perfect information of the game to the child node when descending from the parent node. However, copying the information takes a lot of time in *Pokémon* since the imperfect information is given considerable weight compared to the perfect information at the beginning of the game. Although we tested the algorithms in the simulator which divided the node class and the simulator class, we did not test how calculation cost changed by dividing the node class and the simulator class. This remains as future work.

Most of the uncertainty of *Pokémon* occurs in the simultaneous move nodes when both players choose actions at the same time, and random numbers are generated many times during the game. The *chance node* controls the divergence of the game tree growing for uncertainty and the update of the information sets. Since the chance node is unrelated to the intention of the player, one should not pay attention only to that. In *Pokémon*, chance nodes can successively appear as many as ten times in the game tree, which prevents the player from evaluating the nodes of interest. One solution is to summarize the consecutive chance nodes into one chance node to avoid such a problem. However, the cost to calculate pruning by using this technique is very high since it is necessary to prune branches of the chance nodes frequently depending on the imperfect information of the opponent. This problem occurs when the node class uses the simulator class. It happens when the node class which does not have the perfect information about the game forces the simulator class to prune the number of times that is equal to the number of variations of the imperfect information. However, we need to prune only once if we implement it so that the simulator class which has the perfect information about the game uses the node class when the simulator decides on the moves of the player.

We tested the algorithms using a simulator implemented as explained so far, but we did not inspect how the strength and computational cost of the AI are affected by inverting the master-servant relationship between the node class and the simulator class.

VI. Experiments

A. Compared Algorithms

Cheating MCTS based on Cowling *et al.* [6] (CMCTS) is an algorithm that we used as one of the references to measure performance of ISMCTS. *Cheating* in this experiment means observing the imperfect information of the opponent as perfect information and does not mean violating the rules of the game. We expect that the algorithms are strongly influenced by strategy fusion because of the complexity of the game tree which depends on the imperfect information. Therefore we expect that CMCTS which can observe the imperfect information as perfect information can largely avoid the influence of strategy fusion and will play better than other algorithms.

Cheating Ensemble MCTS (CEMCTS) based on Cowling *et al.* [6] is an extension of CMCTS and plays a game by constructing multiple trees. We equated the number of iterations of the algorithms including CMCTS with the total number of the iterations used for multiple search trees by CEMCTS to keep the fairness in comparing them. Therefore, we expect that each search tree of CEMCTS will become shallower than the search tree of CMCTS. Generally, the player who searches the game tree deeper becomes stronger since he can chose a move by foreseeing the future. Therefore, we expect that CMCTS is stronger than CEMCTS. However, it was confirmed that there is a possibility that the second player of the game plays it pessimistically and becomes weaker if he deeply searches the tree of a game such as Tic-tac-toe, for which the draw or the first player's victory is guaranteed if the first player chooses optimal moves [6]. Thus, we consider that CEMCTS is a worthwhile algorithm to examine since a shallow search might be effective depending on the game.

Determinization MCTS (DMCTS) is a variant of MCTS which uses determinization technique for imperfect information and has been used frequently until ISMCTS was proposed. We expect that DMCTS is not effective for the game in which the imperfect information is important when an algorithm chooses a move, since it is influenced by strategy fusion. We adopted DMCTS as a baseline to see how much ISMCTS can reduce the effect of strategy fusion.

We conducted experiments to compare the performance of four algorithms, namely CMCTS, CEMCTS, DMCTS, and ISMCTS.

B. Evaluation of ISMCTS

While *Pokémon* has various aspects that are worth researching, in this study we focused on comparing the performance of algorithms in a move choice. We used six specific monsters in the experiments and set the values for their parameters based on our knowledge about the game. In addition, we adopted the data from PGL as prior distribution to estimate the imperfect information about the opponent. The number of iterations of each algorithm is 1000. The threshold of the node expanding is 100 in CMCTS, DMCTS, and ISMCTS, and 10 in DMCTS. CEMCTS divided the game tree into ten. In the case of a simultaneous move node of DMCTS, we used EXP3 as the edge choice algorithm in a search tree. In the other cases we used UCB. Each algorithm chose a move which was the highest in the winning rate in the root node. We compared the strength based on 200 battles between every possible pair from the four algorithms.

TABLE I
Winning rates of different algorithms (%)

	CMCTS	CEMCTS	DMCTS	ISMCTS	Average
CMCTS	-	53.5	83.5	75.0	70.7
CEMCTS	46.5	-	73.5	73.5	64.5
DMCTS	16.5	26.5	-	42.5	28.5
ISMCTS	25.0	26.5	57.5	-	36.3

Table I shows that CMCTS, which can observe the imperfect information as perfect information by cheating, is the strongest, as we expected. Information necessary for a player to evaluate his move is imperfect in *Pokémon*, and the difficulty to evaluate the moves is one of the characteristics of *Pokémon*. CMCTS can evaluate the moves more accurately than the other algorithms because it can perfectly know the imperfect information of the opponent. Although some evaluations for moves by CMCTS are not really accurate because of simultaneous moves and uncertainty, it is the same for other algorithms, and we consider that the strength of CMCTS comes from the accuracy of the evaluation of moves.

Table I also shows that CEMCTS is weaker than CMCTS and is stronger than DMCTS and ISMCTS. CEMCTS searches ten different trees in parallel and decides on the next move by averaging the winning probabilities of the trees. Since the total of the computational resources which CEMCTS allows to spend for ten trees is the same as the computational resource which CMCTS spends for a game tree, the tree searching is shallower in CEMCTS than in CMCTS. However, shallow search may sometimes become an advantage since an algorithm that discovered the opponent's surefire way to win may pessimistically evaluate all moves and will not try to do its best to win. We thought that CMCTS did not search pessimistically because there was no surefire way to win in *Pokémon* or the search was too shallow to find it. This diminishes the advantages of CEMCTS while the disadvantage of dispersing the computing resources among ten trees becomes dominant. Because *Pokémon* is a simultaneous move game, it is thought to have no sure procedure to win. On the other hand, DMCTS and ISMCTS that can use ten times more of the computing resources for a single tree lost against CEMCTS, which scattered its computing resources among ten trees. We thought that the advantage of using more computing resources for a single tree has vanished since the imperfect information that DMCTS assumed was different almost every time in 1,000

TABLE II
WINNING RATES WITH DIFFERENT GAME COMPLEXITIES (%)

	3 from 6	3 from 5	3 from 4	3 from 3
CMCTS	70.7	69.5	66.2	65.7
CEMCTS	64.5	63.2	61.2	59.2
DMCTS	28.5	29.2	31.7	34.5
ISMCTS	36.3	38.2	41.0	40.7

iterations. ISMCTS gathered multiple states in one information set regardless of the result of determinization in the first player node, and this can be an advantage of ISMCTS over CEMCTS in terms of computational efficiency. However, the fact that CEMCTS outperformed ISMCTS indicates that the advantage of perfect determinization of CEMCTS exceeds the advantage of ISMCTS.

Table I also shows that ISMCTS is stronger than DMCTS. By replacing the nodes of the search tree with an information set, ISMCTS can avoid dispersion of computing resources in proportion to the number of determinization. Since there are more possibilities for determinization in *Pokémon* than in other games, the aggregating of calculation resources of ISMCTS becomes more effective. In addition, ISMCTS can reduce the influence of strategy fusion caused by determinization.

In conclusion, ISMCTS is stronger than the algorithms that do not cheat, and we assume that it is because ISMCTS can efficiently use computing resources and reduce the influence of strategic fusion in *Pokémon*.

C. Effects of Uncertainty in Games

In the second experiment, we evaluated how winning rates between algorithms are changed when we reduce the complexity of the game tree by reducing possible combinations of parameters of the opponent monsters. In *Pokémon*, players show their own six monsters to each other and select three monsters from them. In the previous experiment, the algorithm knows its own three monsters, but the opponent monsters remain as imperfect information for the algorithm at the beginning of the game. To investigate the effect of the complexity of the game trees on the strengths of the algorithms, we compared the strengths of four algorithms with four different assumptions about how monsters used in the battle are selected: "selecting 3 from 6 monsters", "selecting 3 from 5 monsters", "selecting 3 from 4 monsters", and "selecting 3 from 3 monsters (which means *not selecting*)". The assumptions are used when the algorithms presume the reserve monsters and result in different numbers of determinization. It allows an AI player to guess the three opponent monsters with more certainty if they are selected from fewer monsters. The complexity of the game tree falls if AI can guess the opponent monsters correctly, so the strength of DMCTS will become close to the strength of CMCTS. The numbers of possible determinization at the start of the game by the four selection methods are 10^{16}, $6X10^{15}$, $3X10^{15}$, and 10^{15} respectively.

Table II shows that the winning rates of CMCTS and CEMCTS worsened in comparison with two other algorithms as we decreased the complexity of the game tree, and DMCTS and ISMCTS became stronger. These results show that the complexity of the game tree caused by the imperfect information is related to the strength of the algorithm. The strength of an algorithm depends on the precision of the supposition of the imperfect information.

VII. CONCLUSION AND FUTURE WORK

In this paper, we used *Pokémon* as an imperfect information game with the uncertainty which is more complex than the games studied in previous researches. We evaluated the effectiveness of ISMCTS. The results of the first experiment show that ISMCTS is significantly stronger than DMCTS and is effective for *Pokémon*. The results of the second experiment show that DMCTS and ISMCTS are close in strength to the algorithms that use cheating in simpler games.

It remains to test ISMCTS under more variety of settings, for example, when increasing the number of iterations to search the game tree deeper, testing it with different groups of monsters, and so on. As for the implementation of the simulator, it is worth to examine how the strength and the computing efficiency of the AI will change when we gather the chance nodes and reverse the master-servant relationship of the node class and simulator class. There is room for making the tree search more efficient by gathering different but similar (no need to distinguish) states into a node. Though we only studied in this paper how the AI plays after the game starts, it might be worth studying how to tune the most suitable parameters for each monster, how to select six monsters to form a strong team, and how to choose three monsters from six monsters after knowing the opponent team just before the game starts.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[3] M. L. Ginsberg, "Gib: Imperfect information in a computationally challenging game," *Journal of Artificial Intelligence Research*, vol. 14, pp. 303–358, 2001.

[4] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower bounding Klondike Solitaire with Monte-Carlo planning," in *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009, pp. 26–33.

[5] I. Frank and B. Basin, "Search in games with incomplete information: A case study using Bridge card play," *Artificial Intelligence*, vol. 100, no. 1-2, pp. 87–123, 1998.

[6] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.

[7] 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.

[8] L. Kocsis, "Bandit based Monte-Carlo planning," in *Proceedings of the 17th European Conference on Machine Learning (ECML)*, 2006, pp. 282–293.

[9] 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 (FOCS)*, 1995, pp. 322–331.