Minesweeper: A Reinforcement Learning Approach

Jasmine Bilir
Computer Science
Stanford University
`jbilir@stanford.edu`

Lauren Kong
Computer Science
Stanford University
`lakong@stanford.edu`

Benjamin Wittenbrink
Computer Science
Stanford University
`witten@stanford.edu`

Abstract

Minesweeper is a popular single-player computer game involving sequential decision making with incomplete information of where the safe and mine squares are located on the board. We model this game as a Markov Decision Process and tackle it by taking different approaches: sequential decision planning using online planning through a Monte Carlo Tree Search, and model-free reinforcement learning through Q-learning, SARSA, and a linear approximation of the action value. On a 5×5 board after 30 million iterations, Q-learning performed the best evaluated on a set of random board configurations with a win percentage of just over 31 percent, followed by SARSA at 26, the linear approximation at 25, and MCTS at 15. We believe MCTS suffered largely due to computational restrictions we imposed on the algorithm for feasibility. Nonetheless, all methods substantially outperform a random policy, which has approximately a 0.5 win percentage. Our results demonstrate the success of reinforcement learning strategies in developing optimal Minesweeper action policies and complement the broader literature on reinforcement learning in games.

1 Introduction

Minesweeper is a single player computer game that consists of a board of $n \times m$ clickable squares. All of the squares are initially covered, and some of them contain mines, the number of which is pre-set by the selected difficulty level, with more mines being more difficult. These mines are randomly distributed across the board, and the player only knows how many there are, but not their locations. Every square that does not contain a mine has a number displaying how many adjacent squares contain a mine (a number between 0 and 8). Initially, all the squares are covered, and a player must flag squares that have mines and uncover squares that do not. The player wins by uncovering all squares in the board that do not have mines. If a square containing a mine is uncovered, the player loses the game [12]. While playing, a player is tasked with deciding which square to uncover next, but is only given information about that square by squares adjacent to it. The information provided by the adjacent squares is not guaranteed to be perfect: either the player knows for certain that the square contains a mine, or the player does not have enough information and must randomly choose the next square to uncover. Even if the player plays optimally, they may lose the game due to the imperfect information presented and randomness of board configuration. Due to this, Minesweeper is a Partially Observable Markov Decision Process, but using an appropriate state space, it can be defined and solved as a fully observable Markov Decision Process [8]. In this paper, we model Minesweeper as a Markov Decision Process and explore playing this game using model-free reinforcement learning through Q-learning and SARSA algorithms as well as through online planning via MCTS.

2 Literature Review

Minesweeper was initially studied from a computational complexity lens. Kaye proved the Minesweeper Consistency Problem, that is that the problem of assigning mines to covered tiles, is

NP-complete [4]. Scott, Stege, and Van Rooij expanded on Kaye's findings, arguing that while Kaye proved that Minesweeper may be NP-Complete, his proof failed to conclude that the game is hard. They present an improved model of the problem, called the Minesweeper Inference problem, proving that it is co-NP-complete and thus hard [10].

Nakov and Wei demonstrate that Minesweeper, though a Partially Observable Markov Decision Process (POMDP), can be defined and solved as a fully observable Markov Decision Process (MDP). The authors show that the problem of finding an optimal policy can be solved using modified value iteration. They define a corresponding counting problem to Kaye's minesweeper NP-complete problem of non-contradictory configuration, named $\#MINESWEEPER$, and show it is $\#P$-complete. They explain that computing both the state transition probability and the state value function is equivalent to finding the number of solutions of a system of Pseudo-Boolean equations (or 0-1 Programming), or alternatively, counting the models of a CNF formula describing a minesweeper configuration. They conclude by additionally presenting seven ideas to cut the state space dramatically and evaluate the impact of the free-first-move [8].

In terms of algorithmic approaches, Lordeiro, Haddad, and Cardoso utilize modified Multi-Armed Bandit algorithms to better address certain unique aspects of the Minesweeper game, including the inclusion of human agents and the game's mix of luck and strategy, and show that their approach was successful when deployed on smaller, beginning boards [6]. Mnih et al. apply a novel reinforcement learning technique termed a deep Q-network to solve classic Atari games, leveraging the algorithm's ability to learn successful policies directly from high-dimensional sensory inputs. They conclude that their deep Q-network agent outperformed all previous algorithms employed in the literature, even when given only the pixels and game score as inputs, and achieved a performance level similar to a professional human game tester [7].

In contrast, Sinha et al. propose a new method named as dependents from the independent set using deterministic solution search (DSScsp) to calculate and produce the full set of solutions of a Minesweeper game modeled as a Constraint Satisfaction Problem more efficiently. They also propose a new modified deep Q-learning method, when alternatively modeling the problem as a Markov Decision Process, and achieve higher accuracy and versatile learning. They conclude that this deep Q-learning method, for given mine densities, is the best method to achieve accuracy in Minesweeper games [11]. Similarly, Sajjad applies a neural network based learner to solve Minesweeper, with comparable success rates with common Constraint Satisfaction Problem solvers for Beginner and Intermediate game modes, but arrives at such solutions considerably slower [9].

There has also been recent development in Monte Carlo Tree Search (MCTS) methods. For example, Chaslot et al. explore two progressive strategies for Monte-Carlo Tree Search, progressive bias to use heuristic knowledge to direct the search and progressive unpruning to first reduce the branching far then increase it gradually, to significantly improve the performance of their Go program, combining the strategies for betters success on larger board sizes [3]. Buffet et al. implement a combination of Upper Confidence Bounds for Trees (UTS), a selection mechanism for Monte Carlo Tree Search, and domain specific solvers, to improve its long-term behavior when solving Minesweeper for small and large boards alike, where UTS is commonly used and where variants of CSP are preferred, respectively [2].

3 Model Approach

We modeled the Minesweeper game as a Markov Decision Process (MDP) with the following components. The state space $\mathcal{S}$ consists of all valid configurations of the $n \times m$ board with the specified number of mines k. The state of each cell is either covered or uncovered; if uncovered, it can either have a mine or a number specifying how many mines are in the surrounding squares. Thus, each cell can take on eleven values: covered (which we assign -2), mine (-1), or the number of mines in its neighboring squares (a digit 0 to 8). As a result, clearly an upper bound on the state space is 11^{nm}, which for a simple 5×5 grid exceeds 1e26. However, the majority of these states aren't feasible, i.e., we must have exactly k mines and the cells must be consistent with one another. Nonetheless, the $|\mathcal{S}|$, the number of possible states, still grows exponentially with the board size.

At each stage, the only valid action is to uncover a covered square on the board. Thus, actions are of the form $a = (i, j)$ for $0 \leq i \leq n, 0 \leq j \leq m$. The action space $|\mathcal{A}|$ includes all the uncovered squares on the board at that time.

The transition probability between two states is memory-less, and only depends on the current state and the current board configuration.

A small positive reward (1) is given for landing in a state that is not a mine, but the player has not yet won or lost the game. A large positive reward (10) is given for landing in a state that wins the game, namely all the squares without mines have been successfully uncovered. A large negative reward (-10) is given for landing on a mine, ending the game with the player losing.

4 Methods

Reinforcement learning through model-free methods suit large state and action spaces well because it works directly with the action value function Q, and thus does not require building explicit representation of the transition and reward models like model-based learning does [5]. This is attractive in cases when the problem is high dimensional, including the exponentially large state space of Minesweeper, which is why we implemented model-free Q-learning algorithms, Q-Learning, SARSA, and a linear approximation to tackle the Minesweeper problem.

4.1 Q-Learning

Q-learning applies incremental estimation of the action-value function $Q(s, a)$, and the update is derived from the action value form of the Bellman expectation equation:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q(s', a') \tag{1}$$

where $R(s, a)$ is the reward from taking action a from state s, γ is the discount factor of future rewards, $T(s'|s, a)$ is the transition probability of taking action a from state s' and landing in s' [5]. To eliminate T and R, we can rewrite this as an expectation over samples of reward r and the next state s'. Doing so we obtain the incremental update rule estimate the action value function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2}$$

with r being the immediate reward and α being the learning rate [5]. To guarantee convergence of the action value function, an exploration policy needs to be implemented.

In our implementation, we use an update step $\alpha = 0.1$ and a discount factor $\gamma = 0.9$. For Q, we maintain a dictionary mapping each state to a matrix storing the current estimation of the action value. States are added to the dictionary as they are encountered, as it is computationally infeasible to initialize all states immediately. The exploration policy is defined later in this section.

4.2 SARSA

SARSA, an on-policy reinforcement learning algorithm that is an alternative to Q-learning, uses the actual next action a' to update the action value function and attempts to directly estimate the value of the exploration policy as it follows it. This contrasts from Q-learning, an off-policy method, which maximizes over all possible next actions and attempts to find the value of the optimal policy while following the exploration strategy. SARSA must also be used with an exploration strategy and given the right one, the a' converges to the $\max_{a'}$ of Q-learning. Q-learning and SARSA both converge to an optimal policy, but differ on the speed at which they reach that convergence [5].

The incremental update rule for SARSA is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \tag{3}$$

with the same variables used in Q-learning above [5].

4.3 Linear Action Value Function Approximation

To avoid the computational complexity of the Q lookup table, we also estimate the Q value function using a linear approximation. To do so, we first featurize the state by encoding each cell with a one-hot vector for its current value. Thus, a $n \times m$ board is converted to a $n \times m \times l$ vector, where l corresponds to the maximum number of values each cell could potentially take on. Denote this transformation as f. Then we approximate the value function according to the linear relationship $\tilde{Q} = Wf(s) + B$ where W is a trainable weight matrix (of dimension $(n \times m) \times (n \times m \times l)$) and B is a trainable intercept (of length $n \times m$). We can define the loss of our approximation with respect to the optimal action value function $Q^*(s, a)$ as

$$l(W, B) = \frac{1}{2} \mathop{\mathbb{E}}_{(s,a)\sim\pi^*} \left[(Q^*(s, a) - \tilde{Q}(s, a))^2 \right]. \tag{4}$$

We can minimize this loss using gradient descent to obtain the update rule

$$W \leftarrow W + \alpha(Q^*(s, a) - \tilde{Q}(s, a))\nabla_W \tilde{Q}(s, a). \tag{5}$$

As in Q-learning, we use an approximation of Q^* to obtain the update rule

$$W \leftarrow W + \alpha(r + \gamma \max_{a'} \tilde{Q}(s', a') - \tilde{Q}(s, a))\nabla_W \tilde{Q}(s, a). \tag{6}$$

Note the update rule for B follows analogously. Thus, rather than having to learn action values for every unique state, of which there are exponentially many, in this approach, we only need to learn n^2m^2l parameters. However, this method only works as well as the linear function is able to approximate the value function; for a complex game like Minesweeper, it is likely that this approximation is not sufficient. To address, more complex architectures, like a Deep Q-Network, can be specified, particularly containing non-linear activation functions and more layers.

4.4 Monte Carlo Tree Search

Monte Carlo Tree Search is another online planning strategy that estimates $Q(s, a)$ by running m simulations from the current states. For this reason, it has the advantage of reducing computational complexity over large state spaces. It was because of this advantage that we also choose to include MCTS as an experiment in using reinforcement learning on our minesweeper task.

In each of the m MCTS simulations, we use knowledge of the explored state space, the approximate $Q(s, a)$ and the number of times a state has been selected $N(s, a)$ to explore actions from these states. We update our approximation of $Q(s, a)$ incrementally when observed at a step of the simulation using the equation:

$$Q(s, a) = \frac{(q - Q(s, a))}{N(s, a)} \tag{7}$$

where q is the immediate reward obtained plus the discounted future rewards as observed in the simulation and $N(s, a)$ is incremented by 1 at each update. In the simulation, we follow actions that maximizes the UCB1 exploration heuristic

$$UCB1 = Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a)}} \tag{8}$$

where $N(s) = \Sigma_{(s,a)} N(s, a)$ and c is an exploration parameter. We select actions until a unseen state is reached or a max-depth is reached, ending a simulation. If an unseen state is reached, this state is initialized with $Q(s, a) = 0$ and the necessary $N(s, a)$. In our implementation we are not using any prior knowledge or rollout policy estimation in our initialization of $Q(s, a)$ [5].

In our specific algorithm implementation we modified MCTS for feasibility and comparability such that our max-depth for a simulation was game end (either the player selects a mine and loses or uncovers all the safe squares and wins). We discuss this choice and its costs later in our discussion of results. Other implementation specific include a discount factor of $\gamma = 0.9$ and exploration parameter $c = 100$.

4.5 Exploration

4.5.1 Modified ϵ Greedy

Given the large state space, we devised our methods to heavily encourage exploration. For Q-learning, SARSA, and the action-value approximation, we set the exploration policy as follows:

1. For the first 500,000 games, choose a random action at each step;

2. Thereafter, follow an ϵ-greedy strategy with an initial ϵ of 1, a decay rate of 1e-5, and a minimum ϵ value of 0.01.

This allows the agent to gather information about the state space to inform the action-value function, specifically in areas that would otherwise likely go unexplored with a more conservative ϵ value as they would never be encountered.

4.5.2 Modified UCB1

For Monte Carlo Tree search, since we selected actions based on maximized UCB1 heuristics, we experimented with many different values of our exploration parameter c in order to better optimize for the minesweeper game. Some options we researched and experimented with were UCB1-tuned and UCB1-normalized [1]. When initially running these on small boards and fewer of simulations, we saw little to no difference in their performance. Because of time and compute constraints, we were unable to run these variable c-values on our full-boards and for as many simulations as our full results; however, based on the readings above, it is our hypothesis that UCB1-tuned would have more greatly optimized our exploration/exploitation trade-off. We think this because it would've allowed for more a more tailored approach to exploration, likely to improve results. (We also believe UCB1-tuned would have outperformed UCB1-normalized since we don't believe our value function would have followed a normal distribution.) We were not able to conclude from our experimentation whether UCB1-tuned would have been able to produce results on-par with our modified greedy epsilon that was used on Q-learning and SARSA.

As for experimentation with constant c-values, we tried incremental magnitudes of 10 on the range of 0.1 to 100. Ultimately, for smaller state spaces (i.e. 3×3 boards) and over fewer numbers of games, larger exploration parameters made a slight but not substantive improvement. However, in larger boards and larger state spaces, higher c-values trended with better performance. For this reason, we moved forward with $c = 100$ in our primary experiments, given that this was a sufficiently large and computationally realistic exploration parameter.

5 Results and Analysis

We conducted experiments to assess the performance of Q-learning, SARSA, Action Value Function Approximation, and Monte Carlo Tree Search on the Minesweeper environment. To do so, we developed our own implementations of these methods and configured them for the Minesweeper gym environment. We ran each method for 30 million games and evaluated thier performance using the "out-of-sample" win percentage. In particular, every 100,000 iterations, we stochastically generated 10,000 initial board setups, applied the current optimal (greedy) policy action, and recorded the resulting outcomes. This procedure ensures that we evaluate the success of our method on the full distribution of possible board configuration, rather than just the configurations it was trained on. We also compare our results to the win percentage of a completely random policy.

Due to the exponential complexity of the Minesweeper state space, we limited our attention to a 5×5 grid with 5 mines. This allowed us to evaluate the methods in a reasonable amount of time while still providing meaningful results. The results of our experiments are reported in Figure 1.

As can be seen in Figure 1, all policy-learning methods significantly outperform the random policy (by a multiplier ranging from 30 to 60) and do so almost immediately. Moreover, all of the methods appear to exhibit logarithmic growth as a function of games played: that is, the biggest marginal returns to learning from games played occur in approximately the first three million games. Q-learning achieves the highest win percentage in our experiment, reaching just over 31 percent, while SARSA follows slightly behind with a final win percentage of approximately 26 percent. This ordering is sensible, as we should expect the SARSA policy to be bounded by the Q-learning policy as the next action selected by SARSA is always a possibility for Q-learning, which chooses the optimal action. Interestingly, the linear value approximation ascends incredibly steeply – attaining a win percentage of 22 percent after the first million iterations – and then plateaus, never exceeding 25 percent. This may reflect that the gradient descent optimization got stuck at a local minima (and perhaps an adapative leraning rate should have been implemented). Alternatively, it is possible that this is the extent of the state information that the linear approximation is able to capture.

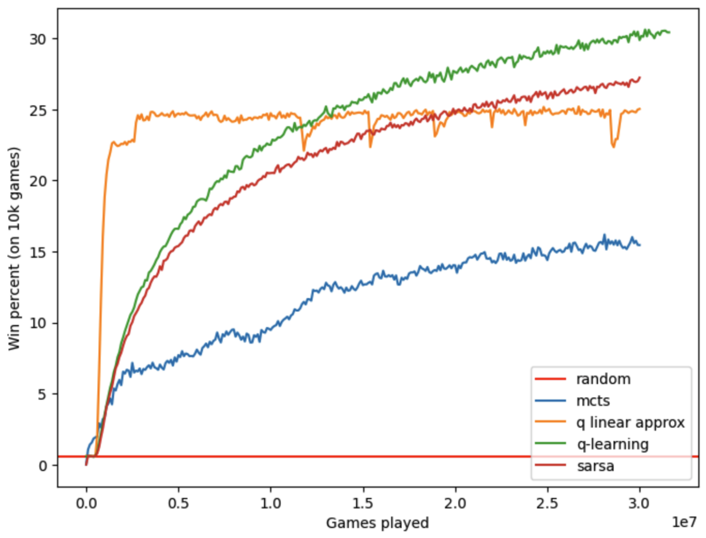

Figure 1: Model performance on 5×5 grid with 5 mines

Finally, the performance of MCTS lags considerably behind that of Q-learning and SARSA. As an exercise, we continued running the MCTS algorithm until it attained a win percentage greater than 25 percent, which took an additional 30 million games. One important note about the implementation of MCTS used for Figure 1 is that for the sake of comparability with the other methods we only ran one simulation for each game state. In particular, we felt that it did not make sense to count MCTS running m simulations for one state as just "one" game and moreover, it would impose significant computational costs to do this over 30 million iterations. Obviously, this inhibits one of the main benefits of MCTS and this likely explains the relatively poor performance depicted above. In Appendix Figure A1, we illustrate the difference this could potentially make, plotting the performance of MCTS with ten simulations per state against MCTS with only one. By the first 400,000 games, MCTS with ten simulations attains a win percentage of slightly over 12 percent, almost six times that of MCTS with only one. However, as discussed above, this requires significant computational power and time, and thus we leave further investigation for future work.

To illustrate the overall state complexity, Appendix Figure A2 shows the number of unique state spaces encountered as a function of games played for Q-learning. By the end of our experimental period, approximately 8,000,000 unique states were stored in the Q-table and the state space was still increasing at the approximate rate of one new state per five games.

6 Conclusion

In this paper, we implemented and evaluated reinforcement learning approaches, such as Q-learning, SARSA, a linear action value approximation, and Monte Carlo Tree Search, on the Minesweeper game. We show that all of these techniques substantially outperform random chance. Compared to one another, Q-learning attains the highest win rate the fastest in our experiments.

Future work should explore the performance of the studied algorithms in larger state spaces. The default "expert" human difficulty for Minesweeper uses a 30×16 grid with 99 mines, which is substantially more complex. In such a setting, methods like pure Q-learning, SARSA, or MCTS will likely struggle, as it is not feasible to explore even a fraction of the possible states. Instead, successful methods will need to take advantage of action value approximation. Here, future research should develop and assess more complex approximation architectures, such as Deep Q Networks (DQN), to evaluate their efficacy in tackling Minesweeper. This line of inquiry will allow us to further refine our understanding of reinforcement learning approaches in this context and to identify methods that yield the most promising results. Ultimately, our work contributes to the growing body of literature on reinforcement learning applications in games and offers valuable insights for researchers interested in harnessing the power of these techniques to solve games.

A Member Contribution

We collaborated well and frequently for this final project, meeting in-person for each stage. We began by researching together ideas of what our respective areas of interest were and what types of algorithms were most compelling to us. We were all excited about Reinforcement Learning and thought the nostalgic game Minesweeper was an interesting application. We met again to begin our algorithm implementation, doing collaborative coding to set up our gym and coding environment and start the Q-learning algorithm with an epsilon-greedy exploration strategy. Jasmine implemented Monte Carlo Tree Search and discussed its exploration strategy, the approach, and results; Lauren conducted the literature review and wrote the abstract, introduction, MDP method, and Q-learning/SARSA approaches; Benjamin implemented the Q-learning, SARSA, linear action value approximation, and Monte Carlo Tree Search methods, wrote up the implementation, results, and conclusions for these methods as well as the linear action value and exploration approach sections. We all edited all sections of the paper. We finished by meeting again, collaborating to finalize the paper and detail our results.

B Code

All code can be found at the following GitHub repository: `https://github.com/benjaminhwittenbrink/cs238project`.

C Figures

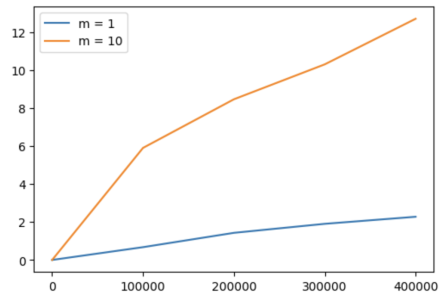

Figure A1: Performance of MCTS over first 400,000 games by m simulatoin steps

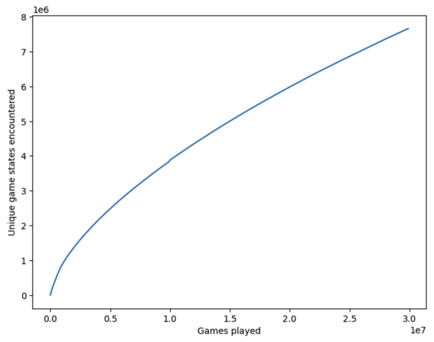

Figure A2: Size of state space by number of games played

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[2] Olivier Buffet, Chang-Shing Lee, Woan-Tyng Lin, and Olivier Teytuad. Optimistic heuristics for minesweeper. In *Advances in Intelligent Systems and Applications-Volume 1: Proceedings of the International Computer Symposium ICS 2012 Held at Hualien, Taiwan, December 12–14, 2012*, pages 199–207. Springer, 2013.

[3] Guillaume M Jb Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[4] Richard Kaye. Minesweeper is np-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.

[5] Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT press, 2022.

[6] Igor Q Lordeiro, Diego B Haddad, and Douglas O Cardoso. Multi-armed bandits for minesweeper: Profiting from exploration–exploitation synergy. *IEEE Transactions on Games*, 14(3):403–412, 2021.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[8] Preslav Nakov and Zile Wei. Minesweeper,# minesweeper. *Unpublished Manuscript, Available at: http://www. minesweeper. info/articles/Minesweeper (Nakov, Wei). pdf*, 2003.

[9] M Hamza Sajjad. Neural network learner for minesweeper. *arXiv preprint arXiv:2212.10446*, 2022.

[10] Allan Scott, Ulrike Stege, and Iris Van Rooij. Minesweeper may not be np-complete but is hard nonetheless. *The Mathematical Intelligencer*, 33:5–17, 2011.

[11] Yash Pratyush Sinha, Pranshu Malviya, and Rupaj Kumar Nayak. Fast constraint satisfaction problem and learning-based algorithm for solving minesweeper. *arXiv preprint arXiv:2105.04120*, 2021.

[12] Wu Tingting, Huang Shunqi, Htung Pa Pa Aung, Mohd Nor Akmal Khalid, and Hiroyuki Iida. Analysis of single-agent game: Case study using minesweeper. In *2020 International Conference on Advanced Information Technologies (ICAIT)*, pages 105–110, 2020.