Review and Analysis of LuckyToilet's MSolver:
A Minesweeper Game-Playing Agent

Abstract
Minesweeper is a one-person game that, at first glance, looks like a very simple game to play. In actuality, this is not the case. The game consists of many different rules that must be followed and understood in order to do well at all. It is because of these rules and the complexity of the game that makes it a great environment for AI testing. Playing the game requires logical, arithmetic, and probabilistic reasoning based on spatial relationships on the board, which is largely the reason why many Minesweeper agents are able to perform better than the average human player. Given the difficulty of hand-crafting strategies to play, AI researchers have always been interested in automatically learning such strategies from experience. Based on the implementation by LuckyToilet, I discuss what the program is actually doing in order to successfully play the game. In addition, I discuss the necessary background knowledge and present the experimental results that they found while testing their implementation, MSolver.

1 Introduction

Minesweeper is a popular oneplayer computer game written by Robert Donner and Curt Johnson. It has been included with Microsoft Windows since 1991. Once a game is initiated, the player is presented with a board containing a map of tiles or squares that are all blank. Hidden among the tiles are mines distributed uniformly at random on the board. This brings the player to the actual task of the game, to uncover all the tiles that do not contain a mine. Throughout the game, the player may do any of 3 actions: mark a tile as a mine, unmark a tile, and uncover a tile. This decision decides whether the player may continue playing, or submit to failure. If the tile does not contain a mine, that tile is uncovered and holds a number representing how many mines are adjacent to it. On the other hand, if the uncovered tile showed a mine, then the player will have blown up the map and failed to complete the puzzle.

At first glance, the rules of Minesweeper make the game look quite simple, but this is a very deceptive view of the game. The rules are easy to understand, but the tactics and strategies that they create through playing the game are much more complex. In order to effectively make decisions while playing, a player requires logic and arithmetic reasoning to get out of most circumstances and game states. In addition, probabilistic reasoning is needed in order to minimize the risk of uncovering a mine when there is no obvious/safe tile to uncover. The game requires a large amount of playtime and in-depth review of past circumstances in order for a

player to create their own strategy. Since hand-crafting playing strategies can prove to be a formidable task, AI researchers have always been interested in automatically learning such strategies from experience. However, this way of learning strategies has not been realized yet, for most of the playing strategies and heuristics used in game playing programs are coded by hand instead of automatically learned.

The remainder of this paper is organized as follows. The next section discusses an array of research advances in separate areas of game-playing artificial intelligence. This gives a great background of where game-playing AI agents currently stand and what has been done to advance a growing form of research. Section 3 describes the approach that MSolver takes in order to perform as it does. Section 4 discusses the results that LuckyToilet witnessed as they tested their program. Finally, Section 5 presents possible areas for future work with the MSolver implementation and Section 6 concludes the paper.

2 Literature Review

Minesweeper has been around for a very long time and has brought quite a bit of attention towards its own complexity and solvability. The world of artificial intelligence has gained knowledge through Minesweeper from research in complexity and solvability in ranging grid dimensions, multi-relational learning, combining myopic optimization and tree search, reactive planning, and the application of Bayesian networks. Even with the vast amount of research done in accordance with Minesweeper rules and specifications, many of the best Minesweeper agents can still only achieve a success rate of 60% to 75%; and these agents are only dealing with games consisting of a small number of mines.

2.1 Computational Complexity

According to the research done by Michiel de Bondt, Minesweeper was found to be PP-hard when the objective is to locate all mines with the highest probability, but is PSPACE-complete when the probability of locating all mines is infinitesimal. In addition, determining the solvability of a Minesweeper game at any point during a game is NP-complete in any grid style, i.e. triangular, square, or hexagonal [3].

Interestingly enough, the work done in [3] shows that it is possible, and relatively simple, to compute the solvability of a Minesweeper map at any point during some play through by way of boolean circuits. This technique may be used in triangular, square, and hexagonal grids after a required initial guess. Seunghoon Lee [4] extended the work done by Bondt by furthering the proof of PPhardness. Lee added new forms of logic circuits which include: NOT, AND, OR gates, a curve, and a splitter for wires. The additional circuits, along with a hexagonal grid style, allowed the addition of uncertainty into the equation, which further showed that Minesweeper is PP-hard.

2.2 Multirelational Learning

An interesting area of study that Minesweeper has been a big part of is the use of a general purpose ILP or multi-relational learning system called Mio [1]. The system is able to play a game many times and learn its own, new strategies to play the game. Shockingly, Mio is able to produce strategies that perform better than the average human players win rate. The user provides the system with background knowledge or premade predicates, which are facts and

rules, or clauses, that explain the games current environment/state. These predicates are used to decipher the game state in order to decide what the best next move is. It is this decision that the system saves and, essentially, learns. Furthermore, these decisions may be replaced by future, more beneficial, decisions or saved and later greedily searched through to find the best outcome [2].

2.3 Search and Optimization

Many agents require reactive planning in order to decide what to do when something happens. Optimization in this part of an agent includes minimizing future possibilities. Sebag and Teytaud [6] implement their own methodology by embedding a non-reactive, or myopic, solver within a consistent reactive planning solver, namely the Upper Confidence-Tree algorithm. The solver is placed in the nodes as a Monte Carlo simulator. The idea is to start off with a reasonably good strategy, and later improving it towards more optimality. This is done with a more greedy, probabilistic search in order to find the best possible strategy.

The search for the most optimal strategy is a very important step that can act quite differently depending on the search algorithm that is used, either approximate or exact. Nakov and Zile [5] created a counting problem called #Minesweeper and explain the differences and advantages in using one of the many search algorithms including the Luby & Velickovic, Inclusion-exclusion, Lozinskii, CDP, and DDP algorithms. Nakov and Zile found that using linear equalities as constraints allowed a fairly fast solution to Minesweeper, but not their proposed #Minesweeper. The use of the DDP algorithm allows them to logically conclude whether there is a mine in a certain spot by computing #Minesweeper twice, once with a mine in the position and the other with the original configuration, and finally taking their ratio.

2.4 Bayesian Networks

In any problem-solving circumstance, a way to model and test a problem is needed in order to understand potential limitations of the given problem. Marta and Jiri utilize the use of Bayesian Network (BN) models with the addition of applying rank-one decompositions (RODs) to conditional probability tables (CPTs) in representing addition. Within the BN model, the computational complexity of probabilistic inference should be enhanced with these additions. Interestingly, the addition had little to no effect on the computational complexity of the BN model. The results point to the possibility that some rank-one decompositions for the CPTs with a local structure are missed, but only when the state of the child variable is observed [7].

3 Approach

The MSolver agent begins by reading the display in order to find the game screen itself. Once the map is found, it picks a random tile to start with and uncovers it. At this point, there is only one tile uncovered (hopefully not a mine), which means it needs to start off by probabilistically finding where the certain number of mines may be. Once the game gets to a point where it can actually figure out safe tiles to uncover, it essentially treats the full game map as many different sections. Each section is only made up of the tiles that relate to it. This makes each section much easier to solve, and allows the agent to keep track of much less. It simulates a divide-and-conquer type of strategy,

which can drastically cut down on memory usage.

4 Results

As previously mentioned, Minesweeper is a strategy dependent game that requires substantial knowledge of logical, arithmetic, and probabilistic reasoning. This is the reason why the average win rate of human players is so low. When implementing an agent to play the game for you, you end up programming yours or somebody elses strategy into the mechanics of the agent itself. This leaves the agent unable to divert from the strategy, and therefore is unable to make mistakes in the sense of human error. The win rate achieved by LuckyToilets MSolver happened to be 50%, which is a quite low statistic, but it has still achieved a greater rate than the average human player.

5 Future Work

The capability to solve a Minesweeper map of LuckyToilets MSolver was shown to win only half the time. A 50% win rate is better than the average human player, but it is still not even close to being perfect. This leaves a large window of opportunity open for other developers to add more complexity and to enhance the efficiency of MSolver. There are plenty of search algorithms that could be implemented in order to test which one is best for this case. Another route for advancing MSolver is to upgrade the way the game map is recognized on the screen. This part of the program needs to work perfectly and quickly in order to allow the agent to solve the puzzle as efficiently and quickly as possible. Lastly, a large part of Minesweeper is the ability to probabilistically guess which tiles to uncover when there are no safe tiles to uncover. The calculations used for guessing the tiles should be revamped and made more accurate in order to allow the agent to decide on the best possible tiles in the current game state.

6 Conclusion

Many of the various papers pertaining to the game of Minesweeper explain an incredible amount of different implementations that are able to perform with completely separate functionalities. Visually, they are solved in the same manner, but the grunt work done by multi-relational learning, myopic optimizations, and multitudes of search algorithms is what allows such a vast, research-ready subject. Minesweeper is an easy-to-understand game, yet quite complex, with rules that are perceived differently by each player/agent, which allows for the creation of many strategies that are still unable to perfectly and logically solve every game state given.

References

[1] L. P. Castillo and S. Wrobel. Multirelational active learning for games. In *Machine Learning Workshop FGML*, volume 1, page 2002. Citeseer, 2002.

[2] L. P. Castillo and S. Wrobel. Learning minesweeper with multirelational learning. In *IJCAI*, pages 533–540, 2003.

[3] M. de Bondt. The computational complexity of minesweeper. *arXiv preprint arXiv:1204.4659*, 2012.

[4] S. Lee. A short note on improved logic circuits in a hexagonal minesweeper. *arXiv preprint arXiv:1602.00398*, 2016.

[5] P. Nakov and Z. Wei. Minesweeper,# minesweeper, 2003.

[6] M. Sebag and O. Teytaud. Combining myopic optimization and tree search: Application to minesweeper. In *LION6, Learning and Intelligent Optimization*, volume 7219, pages 222–236. Sringer Verlag, 2012.

[7] M. Vomlelová and J. Vomlel. Applying bayesian networks in the game of minesweeper. In *Proceedings of Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*, pages 153–162. Citeseer, 2009.