A New Framework to Solve Open Boundary Semeai Problems in the Game of Go

Ching-Nung Lin and Shi-Jim Yen †1

A new framework "Multi-layer Genetic Programming and Teacher Method" is introduced to solve Open Boundary Semeai problems in the game of Go. It has implemented a method to dynamically learn a new algorithm from previous experiences and basic knowledge. In addition, it can memorize what it has learned for the future usage. As a result, Adopting this framework can solve new unknown problems. With the relative time control method, it can provide off-line learning quality and real-time speed. This framework can overcome Monte Carlo Tree Search's disadvantage and work in other domains.

1. Introduction

The open boundary[1] semeai problem is a very difficult problem in Computer Go. Using traditional search methods, it can not be solved because wide branching factors lead to enormous search times. Although Monte Carlo Tree Search(MCTS) is a recent algorithm which dominates Computer Go, semeai is still not solvable with MCTS[2]. This paper introduces a new framework which uses neither traditional nor MCTS methods to solve open boundary semeai problems. This framework does not require any assumptions such as only two chains, one small eye[3], or heuristic boundary settings[1]. It can solve many open boundary semeai problems in reasonable times. This framework uses a new dynamic memorizing and learning algorithm. Instead of MCTS' random simulations, it can reuse what it has learned from previous experiences. Furthermore, it can memorize what it learned and apply that knowledge to solve new problems. Experimental results indicate many hard open boundary problems with 15 to 25 depths from the book[4] can be solved in a single core computer in seconds.

2. Method

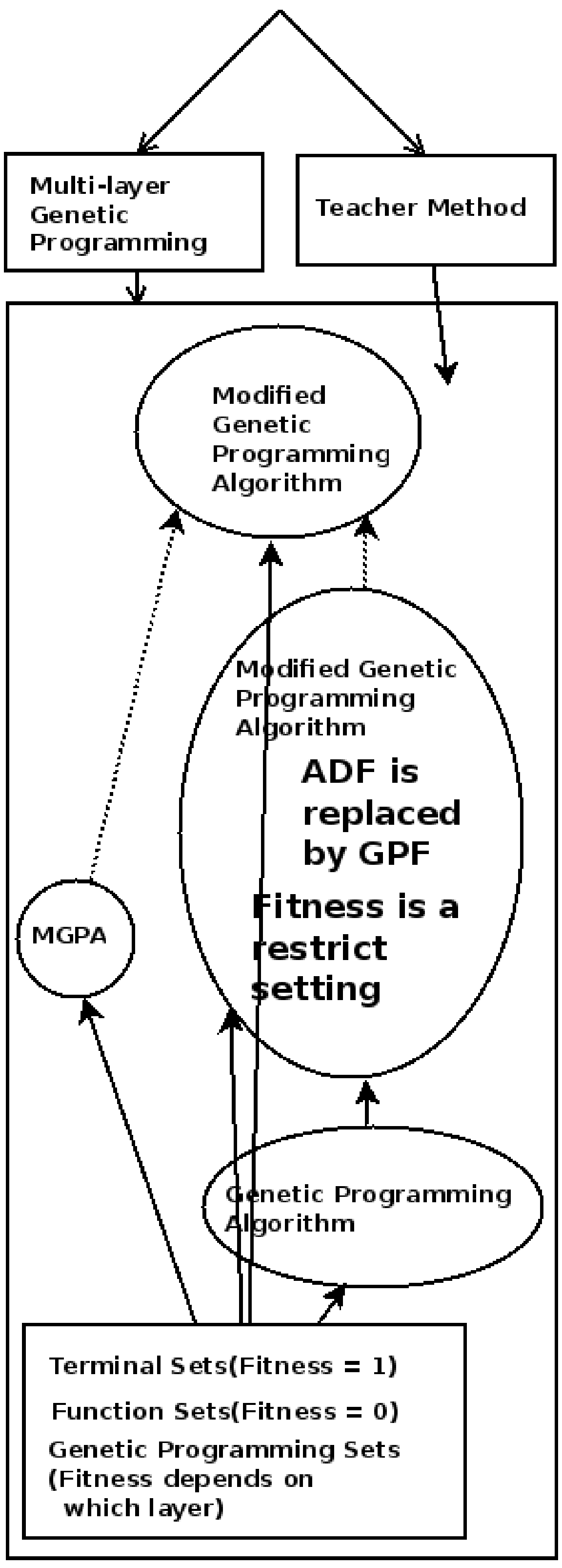
Instead of searching any possible position such as Min-Max search or partial random choosing positions such as MCTS with RAVE, this framework checks the "moves" learned from previous, successfully solved, problems. A "move" is a sequence of different features which are combined from previous learning experiences when solving semeai problems. This might cause the algorithm to grow very complicated during the learning process. Therefore, the memorizing method inside this algorithm will eliminate learning from scratch each time and reduce run-time computing requirements. In addition, this framework implements a method to dynamically modify the knowledge that it has learned and adapt new problems which can overcome the difficulty of implementing exponential heuristic knowledge.

This new framework is named "Multi-layer Genetic Programming and Teacher Method". The model is depicted in Fig. 1.

Teacher Method is heuristic methods, such as to write the best known heuristic solution as a Genetic Programming Function(GPF). For example: Check a Go string liberty = stone1 + check neighbor's emptyspace with a record + neighbor stones + check neighbor's emptyspace with a record + ... This is a correct optimums function. But in most situation, the heuristic methods are unknown. In this case, the teacher method uses Multi-layer Genetic Programming to train with a large problem sets in order to get the GPF. For instance, in the same case to check a Go string liberty, terminal sets are set as stone1, stone2, ... and check neighbor's emptyspace... After training some checking liberty problems, the best fitness function it founds is similar to previous one.

Furthermore, to find a new algorithm by itself, the Multi-layer Genetic Programming is introduced. It is a modified Genetic Programming algorithm[5]. First: The fitness is restricted to set as bigger and shorter are the best. All terminal sets are set with fitness which equals to one. Second: ADF is replaced by GPF. The GPF is a tree generated by Multi-layer Genetic Programming, such as "check a

†1 Dept. of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan
jironglin@gmail.com and sjyen@mail.ndhu.edu.tw



Multi-layer Genetic Programming
Teacher Method
Modified Genetic Programming Algorithm
Modified Genetic Programming Algorithm
ADF is replaced by GPF
Fitness is a restrict setting
MGPA
Genetic Programming Algorithm
Terminal Sets(Fitness = 1)
Function Sets(Fitness = 0)
Genetic Programming Sets (Fitness depends on which layer)

Fig. 1

go string liberty" tree is a GPF.

For instance, to find a ladder solver in Computer Go: Firstly, The base layer uses the normal Genetic Programming Algorithm. One case is to generate a GPF "Check a Go string liberty". The fitness will be like "choose first stone in this string","check the neighbors if it's empty" and so on. The Genetic programming tree is generated as "stone1 + checkneighborspace + stone2 + ...". Secondly, in second layer using Modified Genetic programming algorithm, the function set will be same as first layer, and the first layer GPF is treated as a terminal set with fitness which equals to 2 if it succeeded. Therefore, one of the second layer GPF Atari1 is generated as "Checkliberty(a GFP from the base layer) + iflibertyequal2 + choosethelibertyposition1". Thridly, as the same method, the Atari2, Escape and Capture GPFs in the second layer can be generated. Fourthly, for a ladder solver, it has three layers. Using GPF Atari1, Atari2, Escape and Capture with fitness which equals to 4, it finds a tree to solve all the ladder problems after being trained with some ladder problems. Lastly, Save this tree back to the Genetic Programming Set pool.

The terminal sets are the basic items from the global terminal set pool with fitness which equals to 1 or the advanced GPFs with varied fitness value. For a ladder solver, Atari1, Atari2, Escape and Capture are necessary. Because of fitness which equals to 4, it is easier to generated the correct solution such as Atari1 + Escape + Atari2 + Escape + ... + Capture. The fitness settings for GPFs must set double in each layer. For example, the fitness in base layer is 1; The fitness in second layer is 2; The fitness in third layer three is 4; ... The fitness for global terminal set is assigned to 1 because it provides each set selected opportunely. In order to prevent GPFs from growing exponentially, limiting the depth of the tree can increase the chance to find the optimums solution. The fitness must be set to bigger and shorter are better. In the ladder case, the fitness is known; On the cotrary, most situations are difficult to find the "global optimum" or "correct answer". As a result, considering the bigger fitness in each layer will find the relative best solution tree. When the above layer uses the relative best solution tree(GPF) as a terminal, it considers if it works or not. In addition, the fitness value is recounted which layer it is on, which will solve "choosing the costant factor value" issues.

The Genetic Programming Alogirthm is difficult to find a solution when a lot of steps are required. In this framework, it combines simple procedures to form a complex tree, so a difficult problem can be solved such as Computer Go Semeai problems. For example, Semeai is connected with two groups, and two groups are connected with killing and escaping... After combing simple steps, it can solve a complex task. After this framework gets a new situation such as a new problem, it checks from the base genetic programming node to ver if this new problem is the similar pattern and can be solved with few adjustments. If it doesn't, it keeps checking from bottom to up. Afterwards, if the fitness value is not better than the top layer GPFs, it will combine with previous GPFs and global terminal sets to form a better solution and save it back as a new GPF.

Before this framework, generating a tree to solve all the Computer Go Semeai problem is highly difficult. With this one. All GPFs are adjusting dynamically. This can overcome strong disadvantages of difficulty of implementing exponential heuristic knowledge because it can learn new GFPs by itself with new data feeds in and maintain generalization.

Time control:

Usually Genetic programming is computationally intensive; Therefore, Multi-layer Genetic Programming becomes slow and unpractical. Especially, in the Computer Go, each move needs to be played in few minutes. When the "learning" method is turned off(only GPFs are used), the speed is accepted in realtime applications. This is the "Memory" mechanism. Depending on the time constraits, GFPs can be turned on from the top layer and go down.

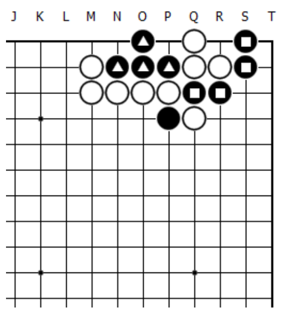


Fig. 2

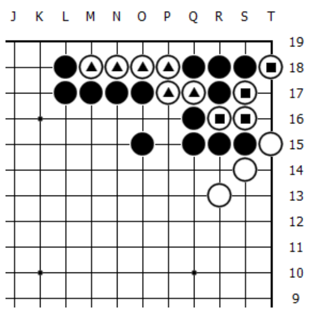


Fig. 3

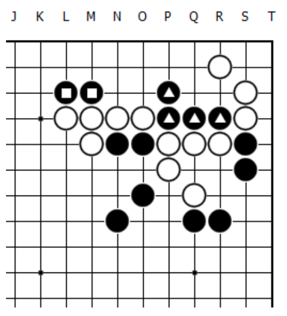


Fig. 4

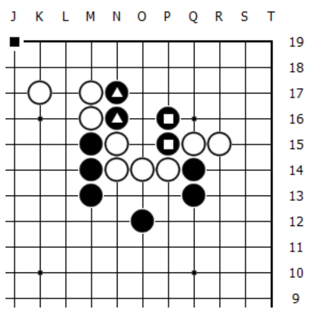


Fig. 5

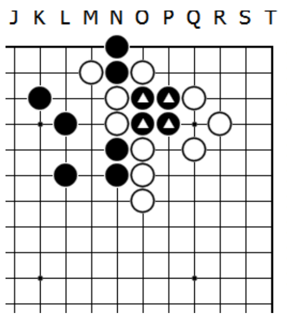


Fig. 6

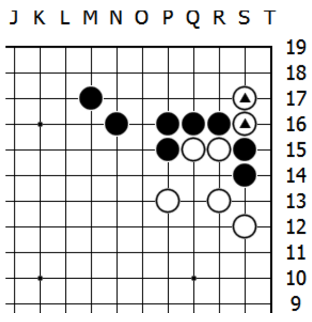


Fig. 7

3. Experimental Result

To kill the four black stones marked with triangles in Figure 2, the next white move needs to interfere with the two black groups marked with squares which leads to more than four group semeai problems. Similarly, to kill the six white stones marked with triangles in Figure 3, the next black move needs to use the four white stones marked with squares. To kill the four black stones marked with triangles in Figure 4, the next white move needs to consider the two black stones marked with squares which leads huge searching nodes if the two groups get connected. To kill either the two black stones marked with triangles or squares in Figure 5 requires more than thirty searching depth, this can not be solved even with millions simulations of Monte-Carlo tree search. To kill black stones marked with triangles in Figure 6 and white stones marked with triangles in Figure 7 requires fifteen to twenty searching depth. In an Intel Core 2 Duo E6550 2.33GHz machine, Setting two semeai groups for the initiation of the algorithm (excluding other heuristic rules) and using our method with single core; Figures 2,3 and 4 can be solved in less than one second; Figure 5,6 and 7 can be solved in two seconds.

Thirty Japanese 3-5 dan open boundary semeai problems were selected from [4]. These problems can be solved within two seconds by our method. The average/minimum/ maximum number of searched nodes are 239.05/29/1398, respectively. Furthermore, by implementing this algorithm to solve new open boundary semeai problems, in most cases, it can adapt well without any algorithm modification; even with a new wider branching factor problem, it can use prior knowledge to create a modified algorithm to solve the problem.

Another advantage for this framework is that the requirement for the runtime memory and the program size is only few kilo bytes. In some experimental tests, it could speed up dramatically when it fits to the L3 cache in a cpu. Likewise, it can run on the mobile phone in an acceptable time.

4. Conclusions

With such high efficiency to solve open boundary semeai problems, this algorithm can be combined with the Monte-Carlo tree search with virtually no penalty. This can overcome MCTS' weaknesses and dramatically reduce unnecessary simulations to concentrate all the computing power on useful play-outs. In the future, solving Seki and open boundary semeai life-death problems will be the next goal. The auto learning and memorizing framework can be used in other domains to deal with complicated problems such as adaptive learning and multi-dimensional job scheduling.

Acknowledgments

The authors thank anonymous reviewers for their valuable comments, and thank the National Science Council of the Republic of China (Taiwan) for financial support of this research under contract numbers NSC 96-2628-E-259-020-MY3 and NSC 98-2221-E-259-021-MY3.

References

1) X. Niu and M. Müller. An open boundary safety-of-territory solver for the game of Go. In J. van den Herik, P. Ciancarini, and H. Donkers, editors, Computer and Games. 5th International Conference, volume 4630 of Lecture Notes in Computer Science, pages 37 - 49, Torino, Italy, 2007. Springer.

2) Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S.; , "A Survey of Monte Carlo Tree Search Methods," Computational Intelligence and AI in Games, IEEE Transactions on , vol.4, no.1, pp.1-43, March 2012

3) Thomas Wolf: A Classification of Semeai with Approach Moves, link on http://lie.math.brocku.ca/twolf/papers/semeai.pdf, 2012

4) 藤沢秀行: 基本手筋事典下,出版社:日本棋院;増補改訂版(1995/07),ISBN-10:4818204021

5) John R. Koza. Genetic programming as a means for programming computers by natural selection. in Stat. Comput. (UK, Journal, vol. 4, pages 191 - 198, 1994