Three Types of Forward Pruning Techniques to Apply the Alpha Beta Algorithm to Turn-Based Strategy Games

Naoyuki Sato
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: satonao@jaist.ac.jp

Kokolo Ikeda
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: kokolo@jaist.ac.jp

***Abstract*—Turn-based strategy games are interesting testbeds for developing artificial players because their rules present developers with several challenges. Currently, Monte-Carlo tree search variants are often utilized to address these challenges. However, we consider it worthwhile introducing minimax search variants with pruning techniques because a turn-based strategy is in some points similar to the games of chess and Shogi, in which minimax variants are known to be effective. Thus, we introduced three forward-pruning techniques to enable us to apply alpha beta search (as a minimax search variant) to turn-based strategy games. This type of search involves fixing unit action orders, generating unit actions selectively, and limiting the number of moving units in a search. We applied our proposed pruning methods by implementing an alpha beta-based artificial player in the Turn-based strategy Academic Package (TUBSTAP) open platform of our institute. This player competed against first- and second-rank players in the TUBSTAP AI competition in 2016. Our proposed player won against the other players in five different maps with an average winning ratio exceeding 70%.**

I. Introduction

Building competitive computer game players is one of the main themes in the field of artificial intelligence. As a result of much research, computer players are sufficiently competent to compete with professional human players in some traditional board games such as chess [1], Shogi (Japanese chess) [2], and Go [3]. On the other hand, there are games in which computer players are weaker than human players. Turn-Based Strategy (TBS) games constitute a genre of games that require additional research to increase the levels of competency of computer players.

TBS games require players to take turns moving their pieces (known as 'units' in TBS) when competing against each other in games such as chess and Shogi. However, TBS games contain rules that complicate the development of strong computer players.

For example, TBS games allow players to manipulate all their pieces (units) in whatever order they prefer in each turn. This rule provides players with a large number of possible actions per turn, which increases the number of nodes and edges for game tree search. Even the use of only six units, each of which has 10 possible actions, result in 720 million possible actions available to a player in a turn. As with other examples, there are various game positions from which players start their games (known as 'maps' in TBS) and unit types have complex relationships to ensure competitiveness in combat. The rules make it difficult to design state evaluation functions with high precision and to apply supervised machine learning, which uses the game records of expert human players.

These difficulties are frequently overcome by adopting Monte-Carlo tree search variants for computer players in TBS games, whereas minimax search variants such as $\alpha\beta$ search are rarely used. However, TBS games have basic game structures similar to those of chess and Shogi, for which $\alpha\beta$ search is known to be effective. In addition, human players often look ahead by considering sequences of consecutive actions to decide their subsequent moves, as is the case with $\alpha\beta$ search, by focusing only on the plausible actions for each position. Therefore, we tried to apply $\alpha\beta$ search to TBS games and evaluated the performance.

We introduced three modifications to decrease the number of edges of the game tree in $\alpha\beta$ search to allow us to increase the depth of the search sufficiently to achieve the desired performance. These modifications are as follows:

- **Fixing the order in which units are allowed to move**
- **Applying selective unit action generation**
- **Limiting the number of moving units in each search**

It is true that these modifications involve some risk of overlooking important or critical moves because these techniques are forward pruning. However, we believe that introducing an appropriate level of this pruning would result in an enhancement of the performance of artificial players in TBS games. Especially, the use of $\alpha\beta$ search in TBS games often does not permit the player to look ahead by even one opponent move (i.e., the search cannot even reach 2 ply deep) in a practical amount of time positions without any pruning, because of the large number of possible actions in one turn. Thus, we expect pruning techniques to have a great effect on the performance, especially if they are capable of increasing the search depth to an extent that would enable them to consider the opponent's future actions.

This paper is organized as follows. In Sect. II, we present work related to this research. Sect. III explains the TUBSTAP

platform we use in this research to evaluate the performance of the proposed method. Sect. IV describes the three types of modifications we adopt in this research. Sect. V to Sect. VII describe preliminary experiments to assess the respective effects of the three modifications. In Sect. VIII, we present the experiments in which our proposed computer player competes against a first- and second-rank player of the TUBSTAP AI competition in 2016. We discuss some future work in Sect. IX.

II. Related Work

A. Minimax Tree Search and Pruning Techniques

Minimax tree search has been widely adopted to develop computer players in several games such as chess and Shogi. Sometimes developers adopted tree searches by removing certain branches of the tree to increase the search depth Such techniques are known as pruning, and pruning techniques are categorized into two types: backward pruning and forward pruning. Backward pruning is pruning that does not affect the search result. The $\alpha\beta$ algorithm is a well-known backward pruning technique [20].

On the other hand, forward pruning removes some game tree branches involving risks to affect the minimax value of the root node. For example, in a classic pruning method shown in a 1960s chess program, tapered n-best search searches [4] only the n best moves, which are decided by some move ordering techniques [5]. These best moves are considered and the other moves are pruned in the search.

More recent techniques often use the alpha/beta values in the $\alpha\beta$ algorithm for forward pruning. For example, futility pruning [6] and null move pruning [7] estimate the upper/lower bounds of evaluation values obtained by moves. In this case, pruning discards moves with evaluation values that have little potential to exceed the alpha value or fall below the beta value. These techniques are used not only for chess but also for Shogi [9], thereby making it possible to search trees deeper than 10 plies.

B. Turn-based Strategy

Considerable research has been performed on TBS games [12] [13] [14] for the "Cid Meier's Civilization" series [10] or its clones. However, we consider the game system of this series to contain many complex factors other than combat by units, such as economy, research, and diplomacy. Thus, we focus on TBS games of which the systems concentrates on combat between units.

The following studies relate to TBS games of this type. A computer player with an evolutionary computational technique is proposed for the Advance Wars clone game [11]. A Monte-Carlo tree search algorithm [15] and upper confidence tree algorithm with fuzzy functions [16] were applied to the open platform TUBSTAP [17]. In this work, we also adopt the TUBSTAP platform for the application of our method.

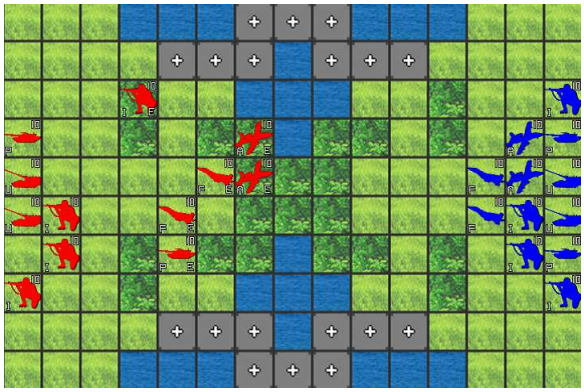

Fig. 1. Screenshot of the TUBSTAP platform.

TABLE I
Unit Relative Strength

Defense / Attack	F	A	P	U	R	I
F	55	65	0	0	0	0
A	0	0	85	115	105	105
P	0	0	55	70	75	75
U	0	0	60	75	65	90
R	70	70	15	50	45	105
I	0	0	5	10	3	55

TABLE II
Land Type - Protective Effect and Movement Cost

Land type	Mountain	Forest	Plain	Road	Sea
	[Protective Effect]				
A, F	0	0	0	0	0
R, I, P, U	0.4	0.3	0.1	0	0
	[Movement Cost]				
A, F	1	1	1	1	1
P, U, R	∞	2	1	1	∞
I	2	1	1	1	∞

III. TUBSTAP Platform

TUBSTAP is an open platform for TBS game computer players. The game system is modeled after "Famicom Wars DS2" [18]. A screenshot of TUBSTAP is displayed in Fig. 1. AI player competitions are held annually using the platform, and the code produced by some of the participants and maps that are used for the competitions are freely available.

We provide an overview of the game rules of the platform. In each turn, each player can manipulate as many units as they like in whatever order the player prefers. Each unit can perform an attack action, a movement action, or an "attack after movement" action in a turn. Attack actions reduce the Hit-Points (HPs) value of the opponent unit at which the attack is directed. When the HP value of a unit is reduced to zero, the unit is excluded from the game. A player wins if the opponent player loses all of their units.

There are six types of units in this platform. That is, Fighter ("F"), Attack Aircraft ("A"), Panzer ("P"), Cannon ("U"), Anti-Air Tank ("R"), and Infantry ("I"). In addition, there are five types of land cells, that is, mountain, forest, plain,

road, and sea. Each unit type and each land type are assigned constant values as shown in Table I and II. The values "relative strength" and "protective effect" define the amount of damage by attack actions. Equation 1 shows the amount of damage, i.e., the amount by which the HP value is reduced by a unit carrying out an attack.

$$\text{damage} = \frac{(\text{relativestrength}) \times (\text{attackHP}) + 70}{100 + (\text{protectiveeffect}) \times (\text{defenseHP})} \quad (1)$$

For example, when a unit A with seven HPs attacks an opponent unit P with nine HPs on a forest cell, the amount of damage unit P experiences is calculated as $\frac{85\times7+70}{100+3\times9}(=5)$.

Cells that can be reached by each unit in turn are defined as "movable capacity" of the unit and the sum of the movement costs the cells along the movement path impose on the unit. The "movable capacity" of F, A, P, U, R, and I is 9, 8, 6, 6, 6, and 3, respectively. The movement cost of each land cell is listed in Table I. A cell is reachable if the sum of the movement costs along the movement path is less than or equal to the unit movable capacity. Each unit cannot proceed through cells occupied by any opponent unit, whereas each unit can pass through cells occupied by friendly units (although, they cannot remain on the same cell).

Each unit except U can attack an opponent unit on an adjacent cell after one movement per turn. Attacks instantly prompt a counter attack by the opponent unit after an attack action. U can only attack more distant units, located in cells found at a Manhattan distance of 2 or 3. These distant attacks do not lead to any counter attack, but U cannot move and attack during the same turn.

This platform serves game systems simpler than those of existing commercial TBS games, although the platform covers essential rules that are found in most TBS games. Thus, we consider this platform as a suitable testbed for this research even though success in creating strong artificial players in this platform does not guarantee success in other TBS games.

IV. Approach

In this section, we explain the three modifications we adopt. We aim to reduce the computational costs for tree search by removing certain edges of the tree. These reductions are important not only because they can limit the computational time to a reasonable amount, but also because they might be able to sufficiently extend the search depth such that it enables them to look ahead at the opponent's actions.

A. Fixing orders of unit to move

In TBS games, the order in which players manipulate their units in a turn is often important. However, there are many cases in which the difference between orders is irrelevant. In addition, there are also cases in which the order of only a few units affects the result (e.g., the case in which the result is affected only by whether a unit X moves before or after another unit Y, even though there are many other units in the position). Therefore, we try to reduce the amount of computation by forcing each player to manipulate their units in some (or a single) fixed order.

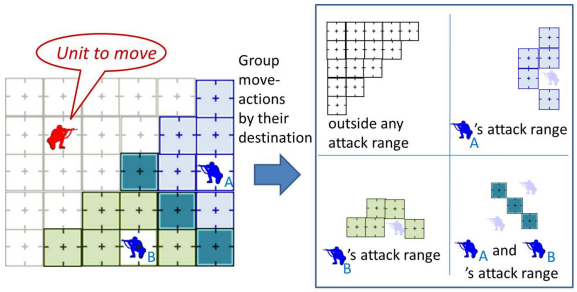

Fig. 2. Grouping movement actions. Only one movement action is generated per group.

In this work we attempted to use the fixed-order patterns listed below for all the units that are assigned IDs(1-N) randomly in a map.

- **Forward** $1, 2, 3, \ldots, N$.
- **Backward** $N, (N-1), (N-2), \ldots, 1$.
- **Cut-Forward** $(\frac{N}{2}+1), (\frac{N}{2}+2), \ldots, N, 1, 2, 3, \ldots, \frac{N}{2}$. Forward order with the former and the latter parts swapped.
- **Cut-Backward** $(\frac{N}{2}-1), (\frac{N}{2}-2), (\frac{N}{2}-3), ..., 1, N, (N-1), ..., \frac{N}{2}$. Backward order with the former and the latter parts swapped.

Depending on the value of a parameter, our proposed player (which appears in the following sections) might consider only one of them, or some of them.

B. Selective unit action generation

At first we provide definitions for some of the specific terms used in this method. Strictly speaking, there are attack actions, movement actions, and "attack after movement" actions in TBS games, although we include "attack after moving" in the attack actions for convenience. Furthermore, we prune movement actions and attack actions differently.

[Movement action]

The number of possible movement actions per unit tends to be large. Thus, we group the movement actions of a unit according to the attack ranges of opponent units. This grouping procedure is illustrated in Fig. 2. Then, we select one movement action from each group, and discard the other actions. This approach seems to be important because other actions in the same group only serve the same opponent units that can attack the unit in the next turn. There are many ways to decide which action to pick from each group; however, we selected actions according to the priorities stated below (the first has higher priority).

1) The extent of protection the unit can take after the movement (the higher the better.)
2) The Manhattan distance after movement of the coordinates from the center of the whole units in the map (the shorter the better.)

If more than one candidate remains, choose only one at random.

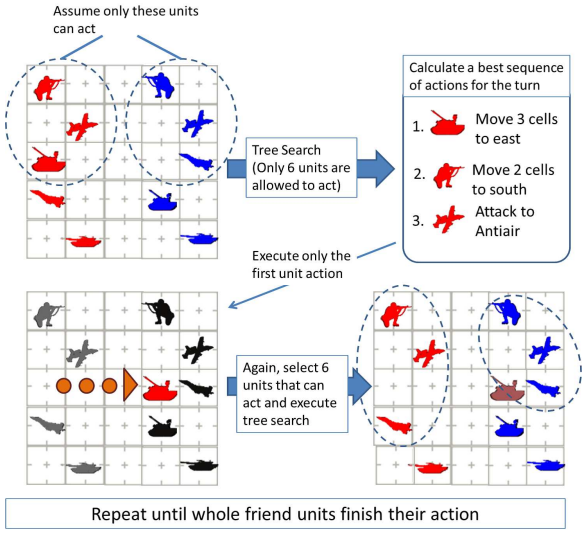

Fig. 3. Tree search with limited number of moving units. Instead of performing a tree search using a large depth, repeat a shallower tree search in which only a limited number of units can move.

[Attack action]

Similarly, we group attack actions to partially prune them. We group the attack actions of a unit according to their target unit (note that there can be more than one attack action from a unit against another unit in a position). Then, we select only one action per group, and discard the others. The action to be chosen is selected by the number of opponent units that can attack the unit in the next turn. The action with the least value is selected. If there are multiple candidates, only one action is selected at random.

C. Limiting the number of moving units

Before we start explaining the detailed issues, to avoid confusion, we provide definitions for words used frequently in this section.

- **Unit action**: Atomic action by a unit. Movement or attack.
- **Player action**: A sequence of multiple **unit actions** that one player can take in one turn. In case a player has N units, their player action consists of N unit actions (unless the player finishes the turn leaving some of their units unmoved).

In TBS games, the number of player actions increases exponentially according to the number of units. Given that a player has N units on their side, each of which can take M possible actions, a minimax tree requires $N!M^N$ leaf nodes to look ahead of whole possible actions of the player in a turn (i.e., search 1 ply deep). This value easily exceeds a reasonable amount for practical use (e.g., $M = 30$ and $N = 6$ result in about 520 billion nodes) as the value of N increases.

Therefore, we try to decrease the number of units involved in a tree search. The procedure is illustrated in Fig. 3. We repeat the two following operations until the whole the unit finishes its action, (a) search the best **player action** by tree search in which only a limited number of units generates possible actions, (b) execute only the first **unit action** from the obtained best player action.

This procedure is able to decrease the exponential factor in the computation. In case each player has N units each of which M possible unit actions can be taken, the amount of nodes needed for a naive D-ply tree search is $(N!M^N)^D$. However, by applying this limitation technique (only N' ($< N$) units can generate actions), the number of nodes becomes fewer than $N(N'!M^{N'})^D$. The number of nodes decreases by a factor of $\frac{1}{N}(\frac{N!}{N'!}M^(N - N'))^D$.

It should be considered how to select such 'movable' units (i.e., units capable of generating possible unit actions in a tree search) in this method. The measurements are as follows (the first has a higher priority):

1) HP value of the unit. A higher value is preferred.
2) Manhattan distance from the coordinates at the center of all the units in the map. A smaller distance is preferred.

By adopting these measurements, we ignore weak units (i.e., units incapable of causing a large amount of damage against opponents and can be destroyed easily), or units further from a hot spot. These units can be considered to be less influential on the battle situation.

V. Preliminary Experiment 1: Fixing Unit Orders

In this section, we describe the experiment that was carried out to determine the influence of the fixing order technique on the performance. We prepared an $\alpha\beta$ search player to which we apply the fixing order technique. (Hereafter we refer to the search player as the 'proposed player' to indicate players we built for experiments) This player has two parameter variables: the search depth of the game tree and the number of fixed-order patterns adopted by the player. The performance of the player is measured through battle trials.

A. Design: Artificial Player

The proposed player's search depth (the number of 'player actions' that need to be looked ahead) is 1 or 2. The unit orders considered by the player are {**Forward**}(see section IV-A), {**Forward**, **Backward**}, {**Forward**, **Backward**, **Cut-Forward**, **Cut-Backward**}, or whole possible orders. The state evaluation function adopted by our proposed player is described in the appendix of this paper.

B. Experimental Setup

The proposed player competes against a naive UCT player (described in [15]) that uses 10,000 playouts per unit action generation. The maps illustrated in Fig.4 to 6 are used, and 200 matches are carried out for each map. The proposed player moves first in 100 games, after which the other player plays first in the next 100 games. A drawn match is counted as a 1/2 win for both players.

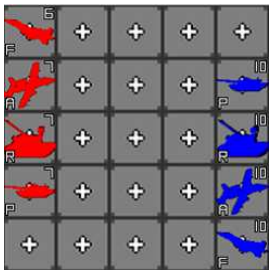

Fig. 4. Map X. Designed by the authors. (F6, A7, R7, P7) for Red player. (F10, A10, R10, P10) for Blue player. Red player move first.

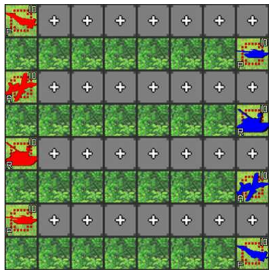

Fig. 5. Map Y. Designed by the authors. (F10, A10, R10, P10) for Red player. (F10, A10, R10, P10) for Blue player. Red player move first.

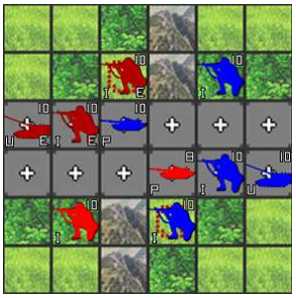

Fig. 6. Map Z. (P8, U10(already enacted), I10(already enacted), I10(already enacted), I10) for Red player. (P10, U10, I10, I10, I10) for Blue player. Red player moves first.

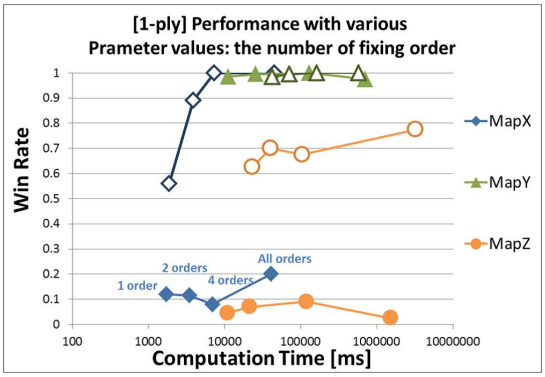

Fig. 7. Win rate against UCT player. 1-ply search. The number of unit orders are varied. Filled/Outlined markers for matches in which proposed player plays first/second.

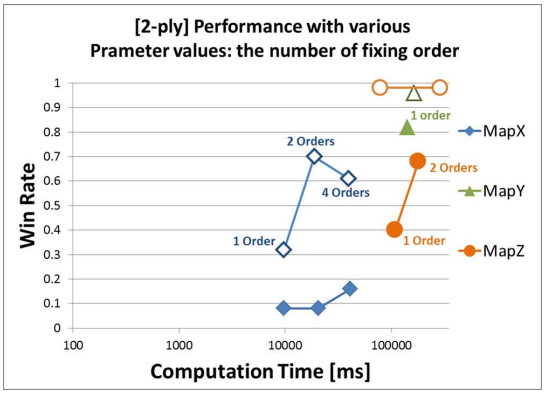

Fig. 8. Win rate against UCT player. 2-ply search. The number of unit orders are varied. Filled/Outlined markers for matches in which the proposed player plays first/second. If the computation time exceeds the time limits (300 seconds), the plot is omitted; e.g., '2 orders' in Map 2 and '4 orders' in Map 3 are omitted because of the time limit.

C. Results

The results of the battle experiments are shown in Fig. 7 and 8. The larger the number of fixed-order patterns is, the higher the performance tends to be. Additionally, the computation time increased according to increasing parameter values. The reduction in the win rate is less than 10% in most cases for the '2 orders' and '4 orders,' whereas the computation times surely decreases by a factor of more than 10 (for '2 orders'). Thus, we conclude that the approach in which the order is fixed is able to improve the efficiency.

On the other hand, there should be possible flaws in our method. In some cases, especially the cases in which we need to eliminate whole opponent units through a narrow path, this pruning method might fail to find an effective move because the correct order in which units are required to move are overlooked.

VI. Preliminary Experiment 2: Selective Action Generation

In this section, we use experiments to verify the influence of the selective action generation approach on performance.

A. Design: Artificial Player

We prepared an $\alpha\beta$ search player, with/without selective action generation. The state evaluation function used by the player is shown in the appendix. Different from Sect. V, this player always adopts whole orders for units to participate in the tree search. We apply selective generation for movement actions, for attack actions or for both of these types of actions and then we observe the performance and the computation time.

B. Experimental Setup

Almost the same settings as in Sect. V are employed. The $\alpha\beta$ search player competes against a naive UCT-based player with 10,000 playouts. The $\alpha\beta$ search player's search depth is either 1 or 2. The options in regard to forward pruning are as follows.

- **Both**: Both the movement actions and attack actions are generated selectively.
- **Move**: Movement actions are generated selectively. Whole attack actions are generated.
- **Attack**: Whole movement actions are generated. Attack actions are generated selectively.
- **No-Prune**: No forward pruning is applied. Whole possible actions are generated.

As in Sect. V, the maps illustrated in Fig. 4 to 6 are used, and the number of matches per each map are also the same.

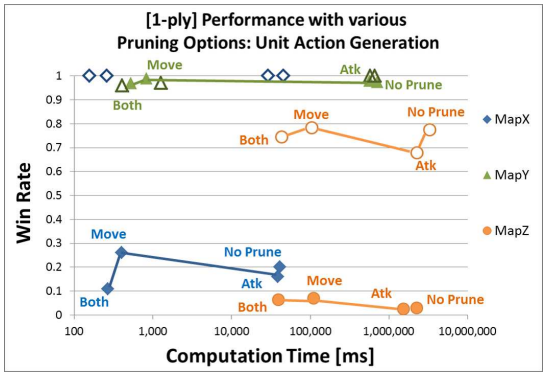

Fig. 9. Win rate against UCT player. 1 ply deep. Pruning options for selective action generation are varied. Filled/Outlined markers for matches in which the proposed player plays first/second.

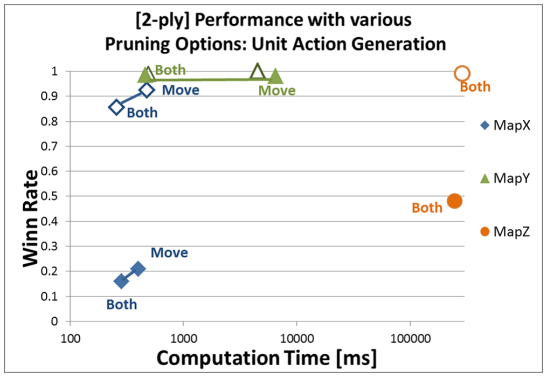

Fig. 10. Win rate against UCT player. 2 ply deep. Pruning options for selective action generation are varied. Filled/Outlined markers for matches in which the proposed player plays first/second. If the computation time exceeds the time limits (300 seconds), the plot is omitted; e.g., each of the 'No-Prune' options are omitted from the plot because their time limits exceed the specified value.

C. Result

The results of the battles are shown in Fig. 9 and 10. The pruning attack actions seem to degrade the performance even though the differences are slight in some cases. On the other hand, pruning movement actions seem not to degrade the performance, although this type of pruning action decreases the computation time significantly by nearly a factor of 10.

Thus, we conclude that the approach involving selective generation is effective in this case, because it does not result in significant performance degradation even though it reduces the computation time.

VII. Preliminary Experiment 3: Limiting the Number of Moving Units

The approach described in Sect. IV-C is examined in this section.

A. Design: Artificial Player

As in the experiments in Sect. V and VI, we prepare an $\alpha\beta$ search player of which the state evaluation function is provided in the appendix. The player employs a tree search with a 1- or 2-ply search in which only the limited number of units can act.

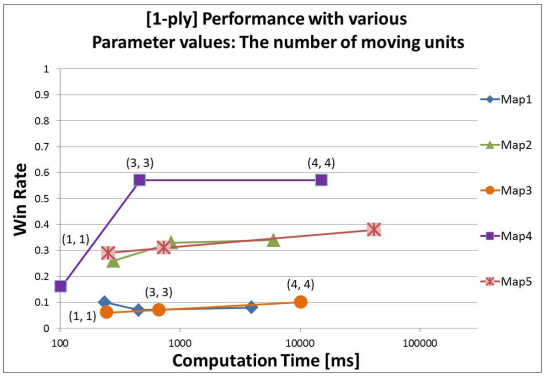

Fig. 11. Win rate against UCT player. 1 ply deep. The parameter value (M, N) represents that "Only M friendly units and only N opponent units can participate in the tree search". Then, (1, 1), (3, 3), and (4, 4) are employed here.

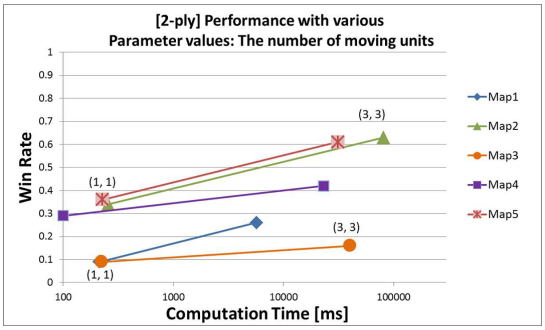

Fig. 12. Win rate against UCT player. 2 ply deep. The parameter value (M, N) represents that "Only M friendly units and only N opponent units can participate in the tree search". Then, (1, 1), (3, 3), and (4, 4) are employed here, although (4, 4) exceeded the time limit (300 seconds).

B. Experimental Setup

The number of friendly and opponent units that can act in the tree search performed by our player, are (1, 1), (3, 3), and (4, 4). A naive UCT player with 10,000 playouts is the opponent player.

We adopted maps different from those used in previous sections, because this pruning technique allows our player to function on larger maps. The five sample maps bundled with TUBSTAP platform ver. 1.07 (available on the website) are used. The number of red/blue units are (6, 6), (8, 4), (7, 7), (5, 5), and (7, 7), respectively. The number of matches per each map is the same as in Sect. V.

C. Results

The results are shown in Fig. 11 to 12. When the parameter values are increased, the winning ratio and computation time are increased. Additionally, the performance improved with the deeper search except for the battles on map 4. The results show that this limiting approach might harm the performance in some cases, but it surely seems to decrease the computation time.

TABLE III
WIN RATES AGAINST FIRST- AND SECOND-RANK AI PLAYERS (200 MATCHES FOR EACH MAP)

% wins against first-rank player "M-UCT" (with 95% C.I.)						
	Map-Fujiki	Map-Ishitobi	Map-Muto	Map-Sato	Map-Takahashi	Averaged
Move First	90 (±5.9)	75 (±8.5)	41 (±9.6)	93 (±5.0)	66 (±9.3)	73 (±3.9)
Move Second	94 (±4.7)	65 (±9.3)	26 (±8.6)	98 (±2.7)	92 (±5.3)	75 (±3.8)
Total	92 (±3.8)	70 (±6.4)	34 (±4.2)	96 (±1.7)	79 (±3.6)	74 (±2.7)
% wins against second-rank player "DLMC-PW55" (with 95% C.I.)						
	Map-Fujiki	Map-Ishitobi	Map-Muto	Map-Sato	Map-Takahashi	Averaged
Move First	89 (±6.1)	65 (±9.3)	82 (±7.5)	97 (±3.3)	63 (±9.5)	79 (±3.6)
Move Second	82 (±7.5)	53 (±9.8)	78 (±8.1)	97 (±3.3)	80 (±7.8)	77 (±3.7)
Total	86 (±4.8)	59 (±4.3)	80 (±3.5)	97 (±1.5)	72 (±3.9)	78 (±2.6)

VIII. EXPERIMENT: PERFORMANCE EVALUATION

Next, we assess the performance of the proposed method against some advanced players. We prepared a player by applying all three the modifications. The proposed player competes against the winners of the TUBSTAP AI competition 2016 [19] we adopted as the opponent players in this experiment.

A. Design: Artificial Player

We prepared an $\alpha\beta$ player by employing the three forward pruning techniques. The parameter values that were used for pruning in the proposed player were decided by experiments that are omitted from this paper (because of space limitations). The parameters are:

- 2-ply deep search.
- Two fixed-order patterns for units' actions. (**Forward** and **Backward** in section V.)
- Each friendly unit generates pruned movement actions and pruned attack actions as its possible actions in the tree search.
- Each opponent unit generates no movement actions and pruned attack actions as its possible actions in the tree search.
- Five friendly units and 10 opponent units generate possible actions in the tree search.

B. Experimental Setup

The proposed player competes against two players on five maps. The detailed conditions are as follows.

- Five maps are used. (These maps are the same as those used in the GAT2016 competition and are available on the website [19].)
- First-rank player "M-UCT" and second-rank player "DLMC-PW55" in the GAT2016 competition as opponent players. (These players are also available on the website.)
- Two-hundred matches per map for each opponent. One player plays first in 100 games, and the other player plays first in the next 100 games.
- A match that ends in a draw is counted as a 1/2 win for both players.
- Each player uses around 10 seconds to carry out its player action.

We also tested a player without any of the three pruning techniques, though, the player cannot search even 1-ply deep within one minite.

C. Results

The result of the experiment is presented in Table III. The proposed player was significantly more competent than the first-rank player on four out of the five maps, and was significantly stronger against the second-rank player on all the maps. Additionally, the averaged win rates exceeded 70 % over the whole game against the first- and second-rank player, respectively. Thus, we conclude that the proposed player outperformed both of the winners of the 2016 competition. We attribute this performance to the success of our proposed methods in extending the search depth without negatively affecting the precision of the tree search to any significant extent.

IX. CONCLUSION AND FUTURE WORK

We proposed three forward-pruning techniques to apply minimax search variants to TBS games. These methods allow artificial players to search quicker and deeper at the risk of overlooking some important moves.

The influence of these techniques on the respective players ' performance was analyzed. These analyses showed that, although these techniques have a slightly harmful effect on performance, they allow the artificial player to search deeper, thereby resulting in the enhancement of the overall performance on the TUBSTAP platform.

Then, we introduced the techniques into an $\alpha\beta$ search with appropriate parameter values and created an artificial player on the TUBSTAP platform. The experimental result showed that the created player significantly outperformed first- and second-rank players in the TUBSTAP AI 2016 competition.

However, we only evaluated our method on a platform of which the rules are simpler than those of existing commercial TBS game titles. On this platform, the benefits of using our method (reduction of the search space) are greater than the disadvantages (risks of overlooking critical moves). The benefits should be assessed in other TBS game environments because they may not be greater than the disadvantages.

Moreover, our methods have room for improvement because they contain unsophisticated architectures. For example, the method for selecting actions from whole legal moves is designed roughly. In addition, our method is presently unable to intentionally support movement actions that guard important friendly units.

ACKNOWLEDGMENT

The authors wish to thank Muto Kohsuke and Fujiki Tsubasa for releasing the AI players used in this paper, 'M-UCT' and 'DLMC-PW55'.

APPENDIX

A. State Evaluate Function

We prepared artificial players by using an $\alpha\beta$ search, and the search method needs a state evaluation function. Here, we explain the state evaluation function adopted over all the experiments in this paper.

A game position is scored as follows:

1) Each unit (owned by the player to move) carries out an attack action that causes the largest damage out of all possible attack actions. (However, these 'possible attack actions' are not precisely suggested. Whether a unit can attack another unit is roughly (and quickly) estimated by obtaining the Manhattan distance between the two units.)
2) Calculate the weighted sum of unit HPs as the evaluation value. The weight values are: 1 for the turn player's infantry units, 4 for the turn player's units except infantry, -1 for the other player's infantry, -4 for the other player's units except infantry.

REFERENCES

[1] M. Campbell, A. J. Hoane and F. Hsu, "Deep blue," *Artificial intelligence*, Vol.134, No.1, pp.57-83, 2002.
[2] K. Hoki and T. Kaneko, "Large-Scale Optimization for Evaluation Functions with Minimax Search," *Journal of Artificial Intelligence Research*, Vol.49, pp.527-568, 2014.
[3] D. Silver, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol.529, No.7587, pp.484-489, 2016.
[4] R. D. Greenblatt, D. E. Eastlake III and S. D. Crocker, "The Greenblatt Chess Program", *Fall Joint Computing* Conf. Procs., pp.801-810, 1967.
[5] chessprogramming - Move Ordering, [Online]. Available: https://chessprogramming.wikispaces.com/Move+Ordering, [Accessed: 2016-04-30].
[6] J. Schaeffer, "Experiments in search and knowledge," Ph.D. Thesis, Department of Computing Science, University of Waterloo, Canada, 1986.
[7] G. M. Adelson-Velskie, V. I. Arlazarov, M. V. Donskoy, "Some methods of controlling the tree search in chess programs," *Artificial Intelligence*, Vol.6, No.4, pp.361-371, 1975.
[8] T. Romstat, An introduction to late move reductions. [Online]. Available: http://www.glaurungchess.com/lmr.html, [Accessed: 30- Apr- 2016].
[9] K. Hoki and M. Muramatsu, "Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and Late Move Reduction (LMR)," *Entertainment Computing*, Vol.3, No.3, pp.51-57, 2012.
[10] Sid Meier's Civilization Beyond Earth, [Online]. Available: https://www.civilization.com/jp/games/civilization-beyond-earth/. [Accessed: 19- Feb- 2016].
[11] HJ. M. Bergsma and P. Spronck, "Adaptive Spatial Reasoning for Turn-based Strategy Games," *AIIDE*, Proc., pp.161-166, 2008.
[12] W. Stefan and I. Watson, "Using reinfocement learning for city site selection in the turn-based strategy game Civilization IV," *CIG*, Proc., pp.372-377, 2008.
[13] C. Amato and G. Shani, "High-level Reinforcement Learning in Strategy Games," *AAMAS*, Proc., pp.75-82, 2010.
[14] P. Ulam, et al., "Using Model-Based Reflection to Guide Reinforcement Learning," *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, Proc., pp.107-112, 2005.
[15] T. Fujiki, K. Ikeda and V. Simin, "A platform for Turn-Based Strategy Games, with a Comparison of Monte-Carlo Algorithms," *CIG*, proc., pp.407-414, 2015.
[16] K. Mutoh, J. Nishino, "Cutoff Using Fuzzy Evaluation on AI of Turn-based Strategy Games," *IPSJ-GPW*, pp.54-60, 2015 (in Japanese).
[17] TUBSTAP. [Online] (in Japanese). Available: http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs_eng/index.htm [Accessed: 1- May- 2016]
[18] Nintendo, Advance Wars: Days of Ruin for Nintendo DS - Nintendo Game Details. [Online]. Available: http://www.nintendo.com/games/detail/nLeg9iJkPgq3fWBcqtpDNWUJ4IvmaQBY, [Accessed: 2- May- 2016].
[19] TUBSTAP Game AI Tournament 2016. [Online] (in Japanese). Available: http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs/competition_menu.htm. [Accessed: 1- May- 2016]
[20] E. D. James and T. P. Hart, "The alpha-beta, heuristic." Maasachusetts Institute of Technology, 1963.