Dual Lambda Searh の高速化に向けて

副田俊介[†] 金子知適[†] 田中哲朗[††]

Dual λ 探索の高速化手法を提案し，実験によって将棋の終盤における手法の有効性を示した．将棋では終盤でも合法手の数が減らないため，選択的に良い手を読む必要があるが，終盤で重要な攻めの速さなどの要素は正確に評価することが難しい．計算機の詰将棋を解く能力は df-pn アルゴリズムに代表される技術の進歩により大きく進歩しているものの，トップレベルの将棋プログラムでさえ単純な間違いを終盤で犯し逆転を許すことが珍しくない．本論文では将棋の終盤で有効である dual λ 探索を高速化する手法を二つ提案した．これら手法を実装し，Gribmergen の問題集から選んだ問題を解かせたところ，問題によっては探索時間を 10 倍程度に減らすことができ，手法の有効性を示すことができた．

Enhancement of Dual Lambda Search

Shunsuke Soeda,[†] Tomoyuki Kaneko[†] and Tetsuro Tanaka[††]

We propose some search enhancement methods for dual λ-search, and conducted experiments to show their effectiveness. Advances in Shogi endgames have been drastic with the introduction of df-pn algorithm in solving checkmate problems. However, even the top level Shogi programs still make simple mistakes in the endgame. In Shogi, the number of legal moves does not decrease toward the end of the game, thus generally, the combination of forward pruning techniques and heuristic evaluation function is used for endgame positions. Even the use of rules and fiction tuned for the endgame cannot avoid the pruning of moves leading to a winning position, overlooking of fatal moves by the opponent. Dual λ-search algorithm proposed by Soeda takes into account of threats by both players, and suits searching Shogi endgames. In this paper, we will propose search enhancement methods to improve the search efficiency of dual λ-search in Shogi endgames. We have implemented these enhancement to our Shogi endgame solver, and conducted experiments on problems chosen from the Grimbergen's test set. For some problems, the enhancement allowed the solver to search up to 11 times faster than without the enhancement.

1. Introduction

The ability of computer programs to solve Shogi checkmate problems has advanced drastically, mainly due to the introduction of df-pn search algorithm[7]. This has enabled computer programs to find a complicated winning sequence in Shogi endgame that even a top level Shogi player could easily miss. However, Shogi endgame in general is still considered to be a difficult problem for computers, and even the top level programs miss an easy winning move, and lose a game where it could have actually won[4].

† Department of General Systems Studies,Graduate School of Arts and Science, The University of Tokyo
†† Information Technology Center, The University of Tokyo

To solve there problems, we have proposed a tactical search algorithm called dual λ-search[8]. In our previous experiments, we have shown dual λ-search can handle Shogi endgame positions efficiently. However, in these experiments, we limited both the attack moves and the defense moves that were generated. For some problems, our program yielded incorrect results due to the forward pruning of defense moves.

In this paper, we show some enhancement to dual λ-search, and show that some Shogi endgame problems could be solved with dual λ-search without limiting the moves generated.

2. Related Work

Tactical searches are efficient because they limit their moves related to the specific goal they han-

dle and only search a part of the game tree. Thus, they are still effective in such games that global search does not work well. λ-search[9], Generalized Threats Search[1], iterative widening[2] were applied to solve capture games or life-or-death problems in Go, and proof number search and df-pn search algorithm were applied to solve checkmate problems in Shogi[7].

Simulation[5] was first proposed to solve effectively positions with useless interposing piece drops in checkmate search, and is also shown to be effective in a solver for the game of Go[6]. Assume that position P is proved, and position Q is a similar position to P. Simulation borrows moves from the proof tree of P and try to find a quick proof for Q.

3. Dual Lambda Search

Dual λ-search is an algorithm for searching a binary valued game tree. It uses passes together with different orders of threat sequences, and it also takes into account threats by both players.

The basic idea of dual λ-search is to reduce the number of nodes read, by reading only moves where the *attacker* can keep giving threats to the *defender*, while the attacker does not get attack backed by a stronger threat by the defender.

To see if a player gives a threat to his opponent, we use passes. The strength of the threat could be represented by how many times the player must have his opponent to make before the player can win.

For example, if the player makes a check move which is not a checkmate, the player can win his opponent if the opponent makes a pass ☆. We call this that the player is giving a threat level of one to his opponent.

To remove slow attacks, we use simulation together with the pass by the attacker. On attacker's turn, we do a pass to see if the defender has a threat. If the attacker has a threat, we use the proof tree of the threat, and for each move by the attacker, try to simulate the proof tree. If the simulation succeeds, it means that move is a slow attack, and thus should not be read.

☆ There are no passes in Shogi, and moves that does not resolve checks are illegal in Shogi

```
/* For the attacker */
attack(Node node, level, p) {
  /* See the value for smaller level */
  if (level > 0) {
    if (node.not_calculated(level - 1, p))
      attack(node, level - 1, p);

    if (node.isSuccess(level - 1, p)) {
      node.setSuccess(level, p);
      return;
    }
  }

  child_nodes = generate_child(node);

  /* Try pass and see if the defender has a threat
   */
  node_pass = generate_child_after_pass(node);
  attack(node_pass, level - 1, opp(p));

  /* If under threat, use simulation to remove
   * slow attacks
   */
  if (node_pass.isSuccess(level - 1, opp(p))) {
    for (i in child_nodes) {
      simulate(i, node_pass);
    }
  }

  for (i in child_nodes) {
    defense(i, level, p);

    if (i.isSuccess(level, p)) {
      node.setSuccess(level, p);
      return;
    }
  }
  //all moves unsuccessful
  node.setFail(level, p);
}
/* For the defender */
defense(node, level, p) {
  /* We first have to determine the value for
   * smaller level
   */
  if (level > 0) {
    defense(node, level - 1, p);
    if (node.isSuccess(level - 1, p)) {
      node.setSuccess(level, p);
      return;
    }
  }

  /* First try pass and see if this is a
   * level position
   */
  node_pass = generate_child_after_pass(node);
  attack(node_pass, level - 1, p);
  if (node_pass.isFail(level - 1, p)) {
    node.setFail(level - 1, p);
    return;
```

```
  }

  /* Now we try all the moves
   */
  child_nodes = generate_child(node);

  for(i in child_nodes) {
    simulate(i, node_pass, p);
    if (i.isSuccess(level, p)) continue;

    attack(i, level, p);
    if (i.isFail(level, p)) {
      node.setFail(level, p);
      return;
    }
  }

  //all moves success for attacker
  node.setSuccess(level, p);
}
```

4. Enhancements to Dual Lambda Search

We propose two enhancements to dual λ-search, and show their effectiveness with problems chosen from Grimbergen's test set[3].

We propose two enhancements to dual λ-search; delay defense threat and propagate proof tree.

4.1 Delay Defense Threat

When the defender makes a threat move to the attacker, the attacker must reply to the threat before he can continue his attack. Most of the time, the threat by the defender is useless, and the attacker can defeat the defender when the defender runs out of threat moves.

Thus it is efficient to first ignore threats by the defender, and only read them after all other moves are proved. We call this *delay defense threat*(DDT) enhancement.

The pseudo code of this enhancements is as follows:

```
/* For the defender */
defense(node, level, p) {
...
  child_nodes = generate_child(node);

  /* First try non-threat moves */
  for(i in child_nodes) {
    if (is_threat(node, level, i) continue;

    simulate(i, node_pass, p);
    if (i.isSuccess(level, p)) continue;

    attack(i, level, p);
    if (i.isFail(level, p)) {
      node.setFail(level, p);
      return;
    }
  }
  /* Now try threat moves */
  for(i in child_nodes) {
    if (!is_threat(node, level, i) continue;

    simulate(i, node_pass, p);
    if (i.isSuccess(level, p)) continue;

    attack(i, level, p);
    if (i.isFail(level, p)) {
      node.setFail(level, p);
      return;
    }
  }
...
}
```

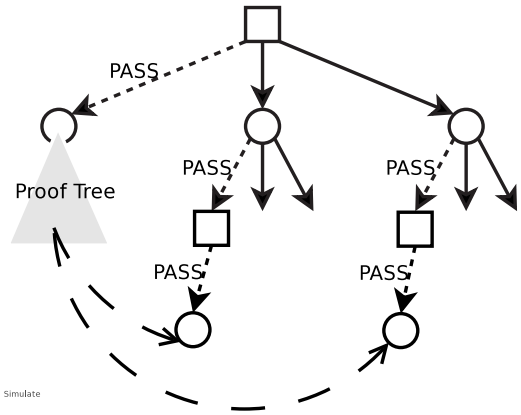

Fig. 1 Use proof tree of threat of the ancestor.

4.2 Propagate Proof Tree

Often a tree proving a threat is also valid for proving threats in its offspring. When trying to prove that the attacker has a threat against the defender, first try to prove it with a proof tree of its ancestor (figure 1). We call this *propagate proof tree*(PPT) enhancement.

4.3 Experimental Results

We have implemented these enhancements to our dual λ-search solver, and chose five problems from the Grimbergen's test set.

Our solver uses df-pn algorithm as the driver for dual λ-search. Our solver generates all moves for defense, but selected moves for attack. These are check moves, moves that increases the mobility of Rooks and Bishops owned by the player, capture moves and moves that move pieces where, on the

Table 1 Number of nodes explored(less is better).

	13	36	37	55	56
None	318863	90406	437024	252735	30372
PPT	318866	90363	424793	252526	30370
DDT	312623	90446	66545	22763	30372
BOTH	312619	90403	67604	22693	30370

next turn, can move to the 25 squares surrounding the opponent King.

The problems chosen were problems 13, 36, 37, 55, 56. The reason these problems were chosen was that their were relatively easy problems to solve.

For problems 37 and 55, the DDT shows great improvement in the number of nodes visited, up to 11 times for problem 55. For problems 13, 36 and 56, the enhancements does not show significant difference in results.

5. Conclusion

We proposed two enhancements to dual λ-search, and conducted experiments for their effectiveness in Shogi endgames. The DDT improved performance of the solver for some problems up to 11 times in the number of nodes explored, but the PPT varied in results, with very little influence.

References

1) T. Cazenave. A generalized threats search algorithm. In *Computers and Games*, Vol. 2883 of *Lecture Notes in Computer Science*, pp. 75–87. Springer, 2002.
2) Tristan Cazenave. Iterative widening. In *IJCAI-01 Proceedings*, Vol. 1, pp. 523–528, 2001.
3) Reijre Grimbergen and Taro Muraoka. What shogi programs still cannot do - a new test set for shogi. In *The 9th Game Programming Workshop in Japan*, pp. 40–47, 2004.
4) Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason. Computer shogi. *Artif. Intell.*, Vol. 134, No. 1-2, pp. 121–144, 2002.
5) Yasuhito Kawano. Using similar positions to search game trees. In *Games of No Chance*, pp. 193–202. Cambridge University Press, 1996.
6) Akihiro Kishimoto and Martin Müller. Df-pn in go: An application to the one-eye problem. In *Advances in Computer Games 10*, pp. 125–141, 2003.
7) Ayumu Nagai and Hiroshi Imai. Application of df-pn algorithm to a program to solve tsume-shogi problems. In *IPSJ Journal*, Vol. 43, pp. 1769–1777, 2002.
8) Shunsuke Soeda, Tomoyuki Kaneko, and Tetsuro Tanaka. Dual lambda search and its application to shogi endgames. In *Advances in Computer Games 11*. Universiteit Maastricht Institute for Knowledge and Agent Technology, 2005.
9) Thomas Thomsen. Lambda-search in game trees — with application to go. *Lecture Notes in Computer Science*, Vol. 2063, pp. 19–38, 2001.