8. Search Depth

Jonathan Schaeffer
jonathan@cs.ualberta.ca
www.cs.ualberta.ca/~jonathan

Search Depth

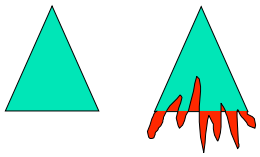
- So far, we have always assumed that all searches are to a fixed depth
- Nice properties in that the search is predictable
  - Can compare search tree sizes
  - Can compare result of search
  - Can predict the result of a search in advance
- But… is this the best way to get the best result?

Investing

- Want to make the best investment of a scarce resource
- You have 1,000 shares and 5 stocks to invest
  - A) 200 shares each?
  - B) Identify likely winners and invest more than 200 shares in them?
  - C) Identify likely losers and invest less than 200 shares in them?

Investing in Search

- Identify sub-trees with good prospects and increase their search depth
- Identify sub-tress with poor prospects and decrease their search depth



Search Reductions

- Identify cases where deeper search is unlikely to be beneficial
- Could do it based on the previous search value ($v << \alpha$), but then you might cut this move off forever from further consideration
- Want an adaptive scheme, automatically discovered from the search

Null Move Search

- For many domains, the option of making a *null move* (passing) is illegal
- However, to make a move that is a no-op for many games would be terrible
- Search the null move case and consider the result a lower bound on what can be achieved
  - Assumes that if you did make a move you could do better than no move at all

Null Move Search

- If we now consider a null move, in addition to all other moves, we have not achieved much
  - Increased the branching factor by 1
- Used a reduced depth for the null move search
- If the reduced depth search can cause a cut-off, then exit the node!

Null Move Search

- Basic idea:
  - If we give the opponent two moves in a row and we still have a great position, then there is no point investing more effort in this analysis

Null Move Search [1]

```
/* Before searching any children…
   /* Make a null move */
   score = -AlphaBeta( s, -beta, -alpha, d-1-r );
   /* Unmake a null move */
   if( score >= beta )
        return( score );
/* Search the children */
```


Null Move Search

- Can do this recursively [2]
  - But don't have a null move follow a null move!
- How do you choose r?
  - Need at least 1 for any savings
  - Is 2 too aggressive?
- Beware of *zugzwang*!

ProbCut [3]

- Another idea for automatically deciding where search effort is unlikely to be beneficial
- Analyse the program's behaviour
  - Calculate the likelihood that the depth *d* search can result in a Δ change in the score
  - Eliminate a node if the score of a depth reduced search is unlikely to return a score necessary to be relevant

Search Extensions

- If there is a line that is interesting, promising, or volatile, maybe the search should be extended
- Identify indicators of "interest" and do a deeper search
- Can use application-dependent knowledge
  - Eg., check moves in chess

Singular Extensions [4]

- Identify forced moves and extend their search an additional move deeper
- One simple definition of forced is one move being "significantly better than all the alternatives"
- Manipulate the alpha-beta windows to identify a forced move

Singular Extensions

- Applying the algorithm at ALL nodes has little overhead
- Applying the algorithm at CUT nodes…
  - Necessary, or you will miss most forced moves by one player
  - Implies continue searching, even though you know a cut-off has occurred
  - This can be a lot of overhead, so you need to be much more aggressive with *r*

Singular Extensions

- Define a forced move as one whose score is at least Δ better than all the alternatives
- Best move has score *v*
- Search remaining moves with a window of (v-Δ, v- Δ +1)
- If any move fails high, revert to a normal search
- If all moves fail low, then a forced move has been found

Singular Extensions

- When a forced move is found, re-search an extra move deeper
- This can happen recursively, resulting in very deep searches
- Store "forced" move property in TT

Search Extensions

- All experiments show that good search extensions/reductions defeat a program without this feature
- Chess
  - Deep Blue team reported 40-move wins found with d=12 searches!
- Checkers
  - Chinook has a nominal search depth of 19, a median position evaluation of 26 and a maximum depth reached of 45!

Conclusions

- Fixed depth search is not the best investment strategy
- Null moves are easy to try and usually are quite effective
- No “simple” search extension idea; most are based on application-dependent knowledge
- For Ataxx, try search reductions first they will probably pay off big!

References

[1] Don Beal. “A Generalized Quiescence Search Algorithm”, *Artificial Intelligence*, vol. 43, no. 1, pp. 85-98, 1990.

[2] Chrilly Donninger. “Null Move and Deep Search: Selective-Search Heuristics for Obtuse Chess Programs”, *ICCA Journal*, vol. 16, no.3, pp. 137-143, 1993.

[3] Michael Buro.”ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm”, *ICCA Journal*, vol. 18, no. 2, pp. 71-76, 1995.

[4] Thomas Anantharaman, Murray Campbell and Feng-hsung Hsu. “Singular Extensions: Adding Selectivity to Brute-Force Searching”, *Artificial Intelligence*, vol. 43, no. 1, pp. 99-109, 1990.