4. Trees and DAGs

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

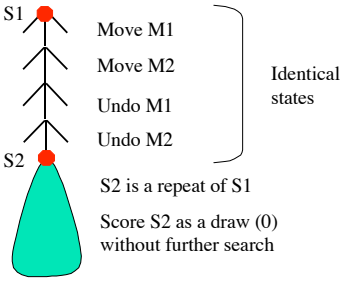
Trees and DAGS

- Many search trees are really search directed acyclic graphs (DAGs)
- Detect cycles and eliminate them
- Detect transpositions and reduce search effort

Cycles

- A path from the root to the current node contains a repeated position
- For most domains, searching a repeated state is unproductive and subsequent search at that node can be eliminated
- Value of a repeated state is application dependent
  - In many games, it is scored as a draw (0)
  - In many domains, it is a non-optimal solution and therefore can be eliminated

Cycles



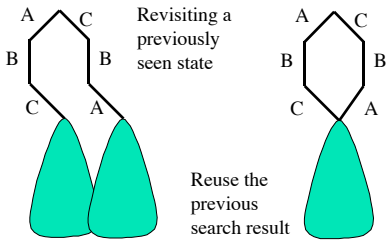
Cycle Detection

- Use a stack
  - Descend - push a state on the stack
  - Ascend - pop a state from the stack
- Enter a node
  - Search the stack for a state equivalent to the current state
  - Optimization is to mark irreversible states and only search as far back as the most recent irreversible state
- Little storage, but can be slow for deep trees

Transpositions

- It may be possible to reach the same state via two different paths
- Want to detect this and (if possible) eliminate the redundant search
- Need to maintain a history of previously seen nodes, not just along the current search path

Transpositions



Transposition Table

- Cache of recently visited positions [1,2]
- Usually implemented as a large hash table (speed)
- Visit a node, search it, and save the result in the TT
- When visiting a node, check if in the TT and if found then…

What if…

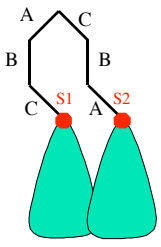
- Position in table and was previously searched *d* ply deep, but current position requires a search to depth *d'*
  - *d'* < *d*: a more accurate result than is needed is available for use!
  - *d'* = *d*: the appropriate accuracy is available
  - *d'* > *d*: the entry is not accurate enough to use

What if…

- How do we interpret the value in the table?
- During a search, 3 results are possible:
  - If V <= α < β, then V is an upper bound on the correct value
  - If α < V < β, then V is an accurate value
  - If α < β <= V, then V is a lower bound on the correct value

This is a concept that causes many difficulties. Make sure you understand it!

Example



Can we use the S1 result for S2?

Accuracy:

S1 and S2 both need to be searched to depth *d*

Value:

S1 searched with a window of (12,20) and returns a value of 5

S1 has an upper bound value of 5

S2 search window is (-10,10)

Thus, S1's result cannot stop the search

Saving in the TT

```
void TTSave( state s; int value;
                    int alpha; int beta; int depth ) {
  if( value <= alpha )
        bound = UPPER;
  else if( value >= beta )
        bound = LOWER;
  else bound = ACCURATE;
  AddToTT( s, value, bound, depth );
}
```

Checking in the TT

```
ptr = TTLookup( state );
if( ptr != NULL && ptr->depth >= d ) {
    if( ptr->bound == LOWER )
        alpha = MAX( alpha, ptr->value );
    if( ptr->bound == UPPER )
        beta = MIN( beta, ptr->value );
    if( ptr->bound == ACCURATE )
        alpha = beta = ptr->value;
    if( alpha >= beta ) /* TT causes a cutoff */
        return( ptr->value );
}
```

Note that the search window can be narrowed, even if a cutoff does not occur!

When to use the TT?

```
int AlphaBeta( state s, int alpha, int beta, int depth ) {
    if( terminal node || depth == 0 ) return( Evaluate( s );
    /* Look in TT before searching */
    ptr = TTLookup( s );
    …
    /* If no cutoff, search */
    …
    /* Save TT result before returning */
    TTSave( s, value, alpha, beta, depth );
    return( value );
}
```

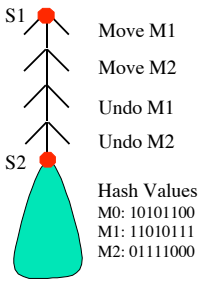

TT Implementation

- Hash table is most common
- Need a fast way to map a state to a hash index
- High performance demands incremental updating of hash index
- Many programs use something similar to Zobrist hashing [3]

Zobrist Hashing

- Table of 64-bit (or more) random numbers for each state feature
- A state transition causes an XORing of the features that change.
- The beauty of the idea is that by XORing, adding a feature and then removing it brings your hash value back to the original value.

Zobrist Example



HashIndex	=	01101011
XOR		11010111
Result	=	10111100
XOR		01111000
Result	=	11000100
XOR		11010111
Result	=	00010011
XOR		01111000
HashIndex	=	01101011

TT Information

- Need to save the following information in the transposition table
  - State
  - Search result
    - Value
    - Bound
    - Accuracy
  - Best move (for use later on)
  - Date

TT State Specification

- Hash table can have collisions, so you need to include the state in an entry
- Storing a complete state may be too much storage and too much computing to compare
- Could use a large random (Zobrist) number instead (at the risk of a small chance of error)

Adding an Entry

- Hash table can become full
- Need a table replacement scheme:
  - Favour deeper searches over shallow ones
  - Favour larger searches over smaller ones
- Popular TT is a two-level table
  - First entry is the best (deepest/largest)
  - Second gives temporal locality [4]

TT Anomalies

- TT may give a more accurate result than needed
  - May result in a solution that normally should not be found
  - But this may depend on the order in which successors are expanded
- Several other subtle points… be wary!

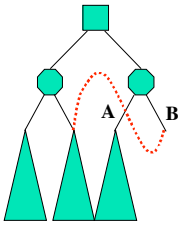
TT Effectiveness

- Application dependent!
- Checkers: roughly a factor of 10
- Chess: roughly a factor of 5
- Othello: roughly a factor of 1.5
- It will vary considerably, depending on the root node, the search depth, and the size of the TT

ETC

- Enhanced transposition cutoff
  - Before searching the "best" move at a node, do a quick search to see if an alternative could achieve the same result but with less effort
- The best move is the one achieving a cutoff, but requiring the least search effort

ETC



Search A then B

What if you had searched B first?

What if the TT entry for B was sufficient to cause a cutoff?

ETC does exploratory work, trying to reuse TT information

ETC Results

- Roughly 25% reduction in the number of nodes examined [5]
- But…
  - Runtime cost of all the additional lookups
  - Application dependent, but roughly a cost of 5%
  - Cost can be reduced by only doing ETC where the benefits are highest -- near the root of the tree

Other Issues

- Bigger tables?
  - More entries sounds good, but less cache locality (an interesting research issue!)
  - If necessary can limit table access to top of tree (where the most benefits are)
- If you use between searches:
  - Time stamp entries so that "old" ones can be overwritten with "new" ones

Conclusions

- Transposition table can be used for:
  - Cycle detection
  - Transpositions
- Look ahead:
  - Move ordering
  - Makes iterative deepening possible
- In general, it is a huge benefit (for many applications, it is the single most important enhancement!)

References

[1] R. Greenblatt, D. Eastlake and S. Crocker. "The Greenblatt Chess Program", Fall Joint Computer Conference, pp. 801-810, 1967. This is the first reference to TTs in the literature.

[2] T.A. Marsland and D. Kopec. "Search", Chapter 30 in *The Computer Science and Engineering Handbook*, A. Tucker (ed.), pp. 676-696, CRC Press, 1997.

[3] A. Zobrist. "A New Hashing Method with Applications for Game Playing", *ICCA Journal*, vol. 13, no. 2, pp. 69-73, 1990.

[4] D. Breuker, J. Uiterwijk, and J. van den Herik. "Replacement Schemes and Two-level Tables", *ICCA Journal*, vol. 19, no. 3, pp. 175-180.

[5] A. Plaat, J. Schaeffer, A. de Bruin and W. Pijls, "Exploiting Graph Properties of Game Trees", AAAI, pp. 234-239, 1996.