6. Evaluation Functions

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

Value of a Leaf Node

- What value do you assign to a leaf node?
- If you have perfect information, then life is easy (leaf node becomes a terminal node)
- Otherwise, need to assign a heuristic value to the node
- Heuristics value must be correlated with the true value
  - The stronger the correlation, the more useful the heuristic

Granularity

- What is the range of possible values for a leaf node?
- If too large?
  - Search may spend too much time trying to improve the value by an insignificant amount
- If too small?
  - Insufficient resolution to differentiate
- In practice, integers are preferred over floating point numbers

Typical Evaluation Function

- Linear sum of features
  - Set of *n* features *f*
  - Each feature has a weight *w*
  - Evaluation = $\Sigma\ w_i \times f_i$
- Non-linear functions can also be a good choice, but tend to be less used in practice
- How do you decide on the features?
- How do you decide on the weights?

Determining the Weights

- Each feature needs a weight
- Marriage proposal evaluation function:
  - F1: spouse's age
  - F2: spouse's pets
  - F3: spouse's clothes
  - F4: spouse's income
- Hopefully, these features are not all equally important!

Determining the Weights

- Traditionally done by hand-tuning
  - Tedious, time-consuming, error-prone
- Many automated techniques proposed and all ineffective until…
- Temporal difference learning! [1]
  - Have the program automatically play itself
  - After each game, modify the weights
  - Make the weights a better predictor of what actually happened

TDLeaf [2]

- Start with an initial set of (random) weights
- Play a game
- Want to modify the weights so that the Move *i* search is a better predictor of Move *i+1* search result
- For each position, find the leaf node of the principal variation (the one responsible for the value at the root).
- Small change to weights so that the position's value is closer to the next search's value

Evaluation Features?

- This is the hard part…
- And it is still mostly black magic
- Ideally would like to automatically discover them, but in practice this is very hard
- GLEM -- Michael Buro presentation [3]

Evaluation Features

- Usually requires an expert to identified features correlated with success
- Can run experiments to verify that a feature is correlated with success
- These features do not need to have anything to do with how humans evaluate a position!

Example: Piece Count

- Human concept of “material balance”
  - Count number of pieces for each side
  - Pieces may have a weighting
- Trivial calculation that is very effective in many games

Example: Mobility

- Mobility -- popular in many games
- Feature value = #moves(me)
  - #moves(you)
- Having more moves to make than the opponent may imply that you have more “freedom” and that can be correlated with success
- No human would ever use such a heuristic, but many human pieces of knowledge are captured by mobility

Example: Square Control

- Control -- popular in many games
- Feature value = #squarescontrolled(me)
  - #squarescontrolled(you)
- “Controlling” a square -- whether actual or just perceived -- may imply that you “own” more of the state that the opponent, and that this can be correlated with success
- Humans use this notion implicitly in some of their knowledge

Warnings!

- The features may not be independent of each other
  - Some characteristic may be explicitly and implicitly over compensated
  - May have to adjust weights or modify features to compensate
- Useful features cover most of the cases
  - Be wary of exceptions
  - Handle the important ones!

Odd/Even Effect

- Iterating one by one depth at a time can cause an unstable search
- Searching to an odd depth can produce an optimistic result (why?)
- Searching to an even depth can produce a pessimistic result (why?)
- Should you be mixing optimistic and pessimistic results?

Odd/Even Effect

- Might see search look like this
  - Depth 4
    - M1 = 20
    - M2 = 25
  - Depth 5
    - M2 = 12
    - M1 = 13
  - Depth 6
    - M1 = 23
    - M2 = 30

Best move keeps changing resulting in a much larger search tree being built

Odd/Even Effect

- Empirical evidence shows that optimistic results generally perform better than pessimistic
- Possible solutions
  - Iterate by 2 at a time
  - Extend search so that only nodes that are at an odd depth can be leaf nodes
  - Modify the evaluation function to make it less depth sensitive

Lose Checkers

- Methodology!?
  - Play a few games and try and understand how you play
  - Is the piece differential important?
  - Is mobility useful?
  - Is control useful?
  - Is the center interesting? The edges?
- Experiment by having your program with feature set 1 play some games against your program with feature set 2

References

[1] Rich Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*, MIT Press, 1998. www-anw.cs.umass.edu/~rich/book/the-book.html

[2] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. “Learning to Play Chess with Temporal Differences”, *Machine Learning*, vol. 40, no. 3, pp. 243-263, 2000.

[3] Michael Buro. “From Simple Features to Sophisticated Evaluation Functions”, *Computers and Games*, Springer-Verlag, LNCS 1558, 1998.