A Generalized Linear Evaluation Model

Michael Buro
Department of Computing Science
University of Alberta

mburo@cs.ualberta.ca
http://www.cs.ualberta.ca/~mburo

Outline

- Evaluation function construction
- GLEM – Building pattern-based evaluations
- Application: Othello
- Future work

Evaluation Function Construction

- EFs are used in **look-ahead seach** to assign **heuristic values** to leaf nodes if no perfect classification is available
- EFs **correlated** with optimization objective. E.g
  - Expected/minimal distance to goal state
  - Probability of winning
    (even in deterministic games? - yes!)
  - Expected payoff
- Classic approach: add weighted features
- Trade-off: **evaluation accuracy vs. speed**

Examples

- Chess: **count pieces - fast!**
  - Material, mobility, King safety, pawn structure ...
  - Add weighted features
    - w(delta-pawns) = 100
    - w(delta-queens) = 990 ...
- Othello: **evaluate parts of the board – fast!**
  - add 51 pre-computed pattern values
- Rubic's Cube: **admissible heuristic**
  - Databases for solving sub-problems
    (lower bound on solution length)

Two Problems

- Where do features come from?
  - Usually provided by **human experts**
  - What if there are **no experts**?
  - What if the expert **can't explain** the feature s/he is using?
  - What if human experts are **weak** in absolute terms?
- How to combine features?
  - Linear, non-linear? What structure?
  - How to assign weights to features?

Search in Function Space : Very Hard!

Genetic Programming

- Breed LISP expressions (trees)
  Atoms refer to state representation or provided features
- Maintain a pool of expressions
- Let the best ones generate offspring
  (“cross-over”, “mutation”)
- Remove weak performers
- Iterate

Hybrid Approach

- Start with (simple) features
  (could be raw state representation)
- Select evaluation model
  (e.g. linear, ANN, decision trees)
- Grow new features by combining previously generated features
- Select new relevant features
- Optimize numerical parameters
- Iterate if not satisfied

GLEM

- Start with **binary features**
  (as simple as “Is a black King on A1?”)
- Grow feature **conjunctions**
- Combine relevant features **linearly**
- Apply monotone **squashing function** to model saturation
- **Optimize feature weights** using linear regression

$$e(p) \;=\; g(\; \sum_i w_i \cdot c_i(p) \;)$$

Conjunctions

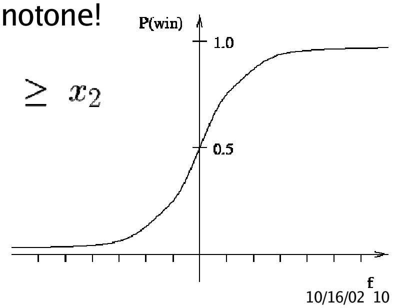
- Complete, can represent perfect evaluation
- **Fast** evaluation
- “only” 2^n feature combinations
- **Natural** non-linear feature interaction. E.g.
  - F_1 : (Black King on 8th rank)
  - F_2 : (White rook on 7th rank)
  - F_1 not correlated with winning
  - F_2 somewhat correlated with winning
  - F_1 & F_2 much more correlated with winning

Top Level: Linear + Squashing

- **Fast** evaluation
- **Efficient** weight **optimization**
  (Gradient based algorithms find global optimum)
- **No** need to apply **squashing function** during game-tree search: monotone!

$$g(x_1) \geq g(x_2) \Leftrightarrow x_1 \geq x_2$$

$$g(x) = \frac{1}{1 + exp(-x)}$$



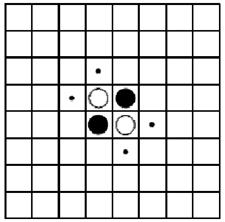
Generating Conjunctions

- **Over-fitting?**
  (good fit on training data, but poor generalization)
- Ad hoc solution: Generate conjunctions that appear at least N times in the training set:
  - **Inductive algorithm**, length 1,2,3...
- Post processing: **remove conjunctions** that are not correlated with winning
- **Future work**:
  - generate maximal conjunctions fast
  - smarter handling of rare conjunctions

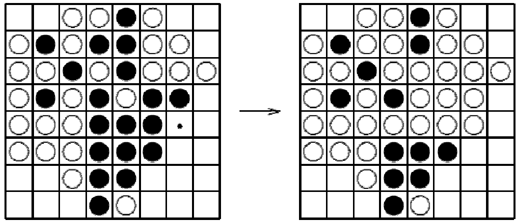
Parameter Optimization

- Generate lots of **training samples**:
  (state, evaluation)
- Generate **conjunctions**
- Solve large (linear) **regression** problem
  - regression takes care of **feature correlation**!
- Boot-strapping: iterate

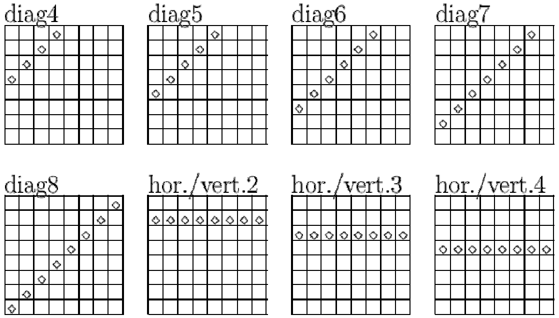
Application: Othello

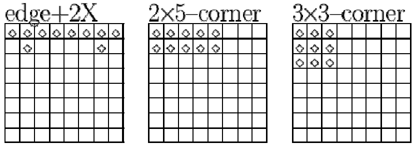


Starting Position (Black To Move)



White To Move

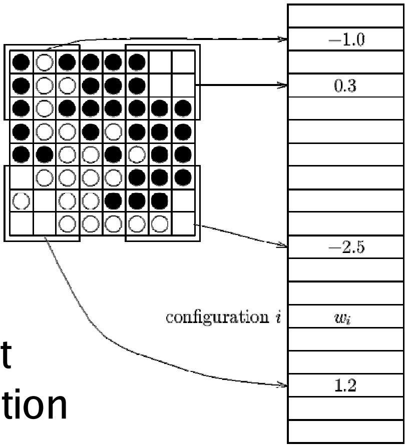




Patterns

3x3-corner

evaluation table



−1.0

0.3

−2.5

configuration i w_i

1.2

Fast Evaluation

$3^9 = 19683$ entries

Logistello's Evaluation Function

- 13 game stages (every 4 discs)
- Sum of 51 precomputed pattern value
  Fast! 1.4 million evaluations/sec on Athlon 1666 MHz
- 1.5 million weights
- 17 million training positions
- Least squares takes 6 hours

Future Work

- Better solution for **rare configurations**
  - Weight bound depending on # of occurrence
- Automated **pattern search**
- Efficient implementation of **large sparse patterns**
- Non-linear top-level combinations
- Other applications: **ataxx**, **backgammon**, **LOA**, **go ...**