Outline

- Genetic algorithms
- Genetic programming
- Human-competitive results
- Games
- Robocode
- Backgammon
- Chess (endgames)
- General discussion

Genetic Algorithms (GA)

- A class of probabilistic optimization algorithms
- Inspired by biological processes: Natural Selection
- Uses analogs of natural selection operators (cf. guy-with-beard)

Genetic Algorithms (cont’d)

- Origins in 1950s
- Consolidated by John Holland, 1975
- Particularly well suited for hard problems, where little is known about the underlying search space
- Widely used in business, science, and engineering
Classes of Search Techniques

- search techniques
  - calculus-based techniques
  - guided random search
  - enumerative techniques
  - Fibonacci
  - sort
  - tabu search
  - hill climbing
  - simulated annealing
  - dynamic programming
  - BFS

- evolutionary algorithms
  - genetic algorithms
  - genetic programming

- BFS

Stochastic Operators

- Fitness value is computed for each individual
- Selection probabilistically selects fittest individuals
- Recombination decomposes two distinct solutions and then randomly mixes their parts to form novel solutions
- Mutation randomly perturbs a candidate solution
Simple Genetic Algorithm

produce initial population of individuals
evaluate fitness of all individuals
while termination-condition not met do
    select fitter individuals for reproduction
    recombine individuals (crossover)
    mutate individuals
    evaluate fitness of modified individuals
end while

The Metaphor

<table>
<thead>
<tr>
<th>Genetic Algorithm</th>
<th>Nature</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimization problem</td>
<td>environment</td>
</tr>
<tr>
<td>feasible solutions</td>
<td>individuals living in environment</td>
</tr>
<tr>
<td>solution quality (fitness function)</td>
<td>individual's degree of adaptation to environment</td>
</tr>
<tr>
<td>set of feasible solutions</td>
<td>population of organisms (species)</td>
</tr>
<tr>
<td>stochastic operators</td>
<td>natural selection, recombination, and mutation</td>
</tr>
<tr>
<td>iteratively applying a set of stochastic operators on a set of feasible solutions</td>
<td>evolution of population(s) to suit their environment</td>
</tr>
</tbody>
</table>

artificial evolution is highly simplified relative to biology
BUT
repeatedly produces surprisingly complex, interesting, and useful solutions
Genetic Programming (GP)

2. Transformed into an effervescent field in large part due to John Koza

Genetic Programming

1. GP = GA, with individuals in population represented as computer programs
2. Usually LISP programs (S-Expressions)
3. Genome composed of functions and terminals
4. GPer determines function set and terminal set
5. Why LISP? Easier (but not easy) to design genetic operators

Genetic Programming

1. Terminal set
2. Function set
3. Fitness measure
4. Control parameters (p<sub>cross</sub>, p<sub>mut</sub>, p<sub>rep</sub>, ...)
5. Termination criterion, result designation

Applying GP

To apply GP, specify:

1. Terminal set
2. Function set
3. Fitness measure
4. Control parameters (p<sub>cross</sub>, p<sub>mut</sub>, p<sub>rep</sub>, ...)
5. Termination criterion, result designation
Example: Symbolic Regression

(Koza)

<table>
<thead>
<tr>
<th>Independent variable X</th>
<th>Dependent variable Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>-0.80</td>
<td>0.84</td>
</tr>
<tr>
<td>-0.60</td>
<td>0.76</td>
</tr>
<tr>
<td>-0.40</td>
<td>0.76</td>
</tr>
<tr>
<td>-0.20</td>
<td>0.84</td>
</tr>
<tr>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.20</td>
<td>1.24</td>
</tr>
<tr>
<td>0.40</td>
<td>1.56</td>
</tr>
<tr>
<td>0.60</td>
<td>1.96</td>
</tr>
<tr>
<td>0.80</td>
<td>2.44</td>
</tr>
<tr>
<td>1.00</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Initial Population

4 randomly created individuals of generation 0

<table>
<thead>
<tr>
<th>Objective:</th>
<th>Find computer program with one input (x) whose output equals given data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Terminal set:</td>
<td>$T = {x, \text{Random-Constants}}$</td>
</tr>
<tr>
<td>2 Function set:</td>
<td>$F = {+, -, *, %}$</td>
</tr>
<tr>
<td>3 Fitness:</td>
<td>$\Sigma \text{abs (program output - given data)}$</td>
</tr>
<tr>
<td>4 Parameters:</td>
<td>Population size $M = 4$</td>
</tr>
<tr>
<td>5 Termination:</td>
<td>Individual emerges with absolute error &lt; 0.1</td>
</tr>
</tbody>
</table>

Symbolic Regression $x^2 + x + 1$

fitness of the 4 individuals in generation 0

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
<td>(d)</td>
</tr>
<tr>
<td>$x + 1$</td>
<td>$x^2 + 1$</td>
<td>2</td>
<td>$x$</td>
</tr>
<tr>
<td>0.67</td>
<td>1.00</td>
<td>1.70</td>
<td>2.67</td>
</tr>
</tbody>
</table>
Symbolic Regression $x^2 + x + 1$

generation 1

Copy of (a)

Mutant of (c)
picking “2” as mutation point

First offspring of crossover of (a) and (b)
picking “+” of parent (a) and left-most “x” of parent (b) as crossover points

Second offspring of crossover of (a) and (b)
picking “+” of parent (a) and left-most “x” of parent (b) as crossover points

Genetic Programming

- Automatically Defined Functions (ADFs)
- Automatically Defined Iterations (ADIs)
- Automatically Defined Recursion (ADR)
- Memory (stacks, queues, lists)
- Architecture-altering operations

Human-Competitive Results

- One aspect of progress in evolutionary computation is increasing generation of “human-competitive” results (www.human-competitive.org):
  - Patent (invention)
  - New scientific result
  - Solution to long-standing problem
  - Wins / Holds its own in regulated competition with human contestant (= live player or human-written program)
Human-Competitive Results

- Over 40 to date
- Examples:
  - Evolved antenna for use by NASA (Lohn, 2004)
  - Automatic quantum computer programming (Spector, 2004)
  - Several analog electronic circuits (Koza et al., mid 90s till today): amplifiers, computational circuits, ...
- MAJOR motivation: As of 2004, yearly contest
- WITH CASH PRIZES!

Games

- Part and parcel of AI since its inception in ’50s.
- 1957: amateur chess (Bernstein)
- 1961: checkers (Samuel)
- 1961-3: learning system for Tic-Tac-Toe (Michie)
- Over the years, many, many games tackled by AIers

Why Study Games?

First, human fascination with game playing is long-standing and pervasive. Anthropologists have catalogued popular games in almost every culture...
Games intrigue us because they address important cognitive functions... The second reason to continue game playing research is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards. (Epstein, 1999)
Why Study Games?

... interactive computer games are the killer application for human-level AI. They are the application that will soon need human-level AI, and they can provide the environments for research on the right kinds of problems that lead to the type of the incremental and integrative research needed to achieve human-level AI.

(Laird and van Lent, 2000)

Robocode

- Written by Mathew Nelson, 2000
- Adopted by IBM (robocode.alphaworks.ibm.com)
- Easy-to-use robotics battle simulator
- Teaching-tool-turned-game-craze
- Different battle types:
  - one-on-one, melee, special
- Different “weight” categories: code size, no. lines

Robocode Player

Java program, Event driven

```java
package topic;
import robocode.*;

public class Walls extends Robot {
    public void run() {
        while (true) {
            ahead(100);
            turnRight(90);
        }
    }

    public void onHitRobot(HitRobotEvent e) {
        back(20);
    }

    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1.0);
    }
}
```
Why Robocode?

- Java programming: popular, accessible
- International league with weekly tournaments (robocode.yajags.com)
  (also robowiki.net/cgi-bin/robowiki?RoboRumble — but no Haiku)
- All players to date human written
- Very little done in the way of machine learning
- Eisenstein 2003: preliminary attempts to evolve player via GP, not very successful

Goal

- Compete in the “real” world, i.e., international league where the (really) real boys play
- Highly sophisticated competitors
- “Real-life” environment

Applying GP

- Tree genome implementing sub-programs
- Main
  - Handling of specific events
    - onScannedRobot
    - onHitWall
    - onHitRobot
- Main functions
  - Move
  - TurnTank
  - TurnGun
  - TurnRadar
  - Fire
Robocode Player’s Code Layout

```java
while (true)
    TurnGunRight(INFINITY); //main code loop
...
OnScannedRobot() {
    MoveTank(<GP#1>);
    TurnTankRight(<GP#2>);
    TurnGunRight(<GP#3>);
}
```

GP Scheme

- **Functions & Terminals:**
  - Mathematical functions: add, sin, abs, ...
  - Numerical constants: constant, random
  - Battle measures: energy, enemyBearing, ...
- **Genotype (=tree) to phenotype (=tank) mapping:**
  - Tree ↦ LISP
  - LISP ↦ Java
  - Embedding of code snippets in player template
  - Compilation into bytecodes

Genome Summary

<table>
<thead>
<tr>
<th>Game-status indicators</th>
<th>Arithmetic and logic functions</th>
<th>Numerical constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy()</td>
<td>Add(x, y)</td>
<td>ERC</td>
</tr>
<tr>
<td>Heading()</td>
<td>Sub(x, y)</td>
<td>Random()</td>
</tr>
<tr>
<td>X()</td>
<td>Mul(x, y)</td>
<td>Zero()</td>
</tr>
<tr>
<td>Y()</td>
<td>Div(x, y)</td>
<td></td>
</tr>
<tr>
<td>MaxX()</td>
<td>Abs(x)</td>
<td></td>
</tr>
<tr>
<td>MaxY()</td>
<td>Neg(x)</td>
<td></td>
</tr>
<tr>
<td>EnemyBearing()</td>
<td>Sin(x)</td>
<td></td>
</tr>
<tr>
<td>EnemyDistance()</td>
<td>Cos(x)</td>
<td></td>
</tr>
<tr>
<td>EnemyVelocity()</td>
<td>ArcSin(x)</td>
<td></td>
</tr>
<tr>
<td>EnemyHeading()</td>
<td>ArcCos(x)</td>
<td></td>
</tr>
<tr>
<td>EnemyEnergy()</td>
<td>IfGreater(x, y, exp_1, exp_2)</td>
<td></td>
</tr>
<tr>
<td>WallDistance()</td>
<td>IfPositive(x, exp_1, exp_2)</td>
<td></td>
</tr>
</tbody>
</table>

Sample Program

```
```
if greater
    wall distance + 50 neg
    enemy bearing random wall bearing
```
Genotype to Phenotype

Evolution: The Rest
- Tournament selection
- Standard crossover, mutation
- Elitism (not much success, probably due to nondeterministic nature of game)
- Experiment with population size, generation count
- Generation zero “grow” method
- Bloat control
- Used Sean Luke’s ECJ package

Evolution: Fitness
- External opponents vs. Coevolution (former proved better)
- Nondeterministic: Num’ rounds in battle has significant effect
- Relative scoring ($S_p$: player, $S_A$: adversary):
  \[ F = \frac{c + S_p}{c + S_p + S_A} \]

HaikuBots
- Code limited to four lines of any length
- Good for GP, which produces much junk code
- Best robot pitted in international league (robocode.yajags.com)
package gep.haiku;import robocode.*;

/**
 * Ver 1.0
 * All code sections on onScannedRobot were evolutionary evolved. No manual
 * optimization was made.
 * 
 * [geep 21/9/2004]
 */
public class GPBotC extends AdvancedRobot {
  public void run() {
    while (true) {
      turnGunRightRadians(Double.POSITIVE_INFINITY);  
    }
  }

  public void onScannedRobot(ScannedRobotEvent e) {
    setTurnRightRadians(robocode.util.Utils.normalRelativeAngle((0 - (Math.abs(Math.sin(Math.abs(getBattleFieldWidth()) + Math.abs(getBattleFieldWidth() + Math.sin((Math.sin(getEnergy()) > 0 ? e.getEnergy() : getY()))) - (setFireBullet((Math.abs(e.getEnergy()) > 0 ? e.getEnergy() : getEnergy()))) == null ? 0.0 : 1.0)) == null ? 0.0 : 1.0))));
  }
}

All Other 27 Players: Human Written!

Evolving Backgammon Players with GP

**Goal:** Evolve a good flat evaluator

1. Start with current board and dice
2. Generate all possible next boards
3. Each board evaluated by GP individual
4. Board with highest score is selected

**Program Architecture**

- Each individual includes two trees:
  - Contact Tree: includes various general and specific board-query functions
  - Race Tree: Much simpler, includes only functions that examine the checkers' positions

**Something About Backgammon**

- Game has two main stages:
  - Contact stage: The two players can hit each other
  - Race stage: No contact between the players

**A Note About Types**

- Use Strongly Typed Genetic Programming (STGP)
- Extension of GP that adds types (Montana, 1995)
- Each node has a return type and argument types
- Node $n_1$ can have child node $n_2$ iff $n_1$ argument type is compatible with $n_2$ return type
- Types: $\text{atomic} = \text{symbol}$, set of atomic types
- Need to define type constraints for backgammon:
  - $\text{atomic} : \text{Float (F)}, \text{Boolean (B)}$
  - $\text{set}: \text{Query (Q)}, \text{contains Float & Boolean}$
Terminal Set for Contact Tree

• Three types of terminals:
  1. Constants
  2. Functions providing information about specific board positions
  3. Functions providing information about the board as a whole

Constants

• An ephemeral random constant (ERC) produces a constant real number in the range \([0,5]\):

\[ F = \text{Float-ERC} \]

Specific Queries

• Internal board positions are numbered 1-24
• Bar marked 0
• Off-board position marked 25

Specific Queries (cont’d)

• When initialized, a random integer \(n\) in range \([0,25]\) is selected (ERC)
• Returns property at position \(n\)
• Specific properties include:
  - \(\text{Player-Exposed}(n)\)
  - \(\text{Player-Blocked}(n)\)
  - \(\text{Player-Tower}(n)\)
  - \(\text{Enemy-Exposed}(n)\)
  - \(\text{Enemy-Blocked}(n)\)

\[ Q = \text{Specific-Property}(n) \]
**General Board Queries**

- Provide general information about the board configuration
- General properties include:
  - Player-Pip
  - Enemy-Pip
  - Total-Hit-Prob
  - Player-Escape
  - Enemy-Escape

\[ Q = \text{General-Property} \]

**Terminal Set for Race Tree**

- Much simpler and contains the functions:

\[ F = \text{Float-ERC} \]
\[ Q = \text{Player-Position(n)} \]

**Function Set**

- Contains arithmetic and logic operators
- Common to both race and contact trees

**Function Set (cont'd)**

- Arithmetic functions:
  \[ F = \text{Add}(F,F) \quad F = \text{Sub}(F,F) \quad F = \text{Mul}(F,F) \]
- Conditional functions:
  \[ F = \text{If}(B,F,F) \]
- Compare functions:
  \[ B \geq (F,F) \quad B \leq (F,F) \]
- Logic function:
  \[ B = \text{And}(B,B) \quad B = \text{Or}(B,B) \quad B = \text{Not}(B) \]
**Genome Summary**

<table>
<thead>
<tr>
<th>Terminal Set:</th>
<th>Terminal Set:</th>
<th>Function Set:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Tree</td>
<td>Race Tree</td>
<td>Both Trees</td>
</tr>
<tr>
<td>F = Float-ERC</td>
<td>F = Float-ERC</td>
<td>F = Add(F, F)</td>
</tr>
<tr>
<td>Q = Player-Exposed(n)</td>
<td>Q = Player-Exposed(n)</td>
<td>F = Sub(F, F)</td>
</tr>
<tr>
<td>Q = Player-Blocked(n)</td>
<td>Q = Player-Blocked(n)</td>
<td>F = Mul(F, F)</td>
</tr>
<tr>
<td>Q = Player-Tower(n)</td>
<td>Q = Player-Tower(n)</td>
<td>F = If(B, F, F)</td>
</tr>
<tr>
<td>Q = Enemy-Exposed(n)</td>
<td>Q = Enemy-Exposed(n)</td>
<td>B = Greater(F, F)</td>
</tr>
<tr>
<td>Q = Enemy-Blocked(n)</td>
<td>Q = Enemy-Blocked(n)</td>
<td>B = Smaller(F, F)</td>
</tr>
<tr>
<td>F = Enemy-Pip</td>
<td>F = Enemy-Pip</td>
<td>B = And(B, B)</td>
</tr>
<tr>
<td>F = Total-Hit-Prob</td>
<td>F = Total-Hit-Prob</td>
<td>B = Or(B, B)</td>
</tr>
<tr>
<td>F = Player-Escape</td>
<td>F = Player-Escape</td>
<td>B = Not(B)</td>
</tr>
</tbody>
</table>

**Fitness Measure**

- Uses external opponent to evaluate individuals
- Each individual plays a 100-game tournament vs. Pubeval
- Fitness of individual (i=individual, p=pubeval):
  \[ F = \frac{Si}{Si + Sp} \]

**First Approach: External Opponent**

- The external opponent *Pubeval* (Tesauro, 1993), is used as a "teacher"
- Individual's fitness determined by playing against teacher
Control Parameters

- Major parameters:
  - Population size ($M$): 128
  - Number of generations ($G$): 500

- Minor parameters:
  - Probability of selecting genetic operator (reproduction, crossover, mutation)
  - Selection operator
  - Methods of growing trees

Termination Criterion & Result Designation

- Termination: 500 generations
- Result Designation:
  - Every 5 generations, the 4 best individuals play a 1000-game tournament vs. Pubeval
  - Individual with best score over entire run is declared best-of-run

Results (External Opponent)

- Benchmark:
  - Results look great, but keep in mind that fitness is over 100 games played
  - So, we applied a 1000-game tournament vs. Pubeval every 5 generations
Coevolution

• Results obtained using external opponent approach are good — but we wanted Better. Stronger. Faster.
  • Suspected to be over-fitted
  • Apply coevolution: Individuals play against each other
  • Use Single-Elimination Tournament (Angeline et al. 1993)

Benchmark (External Opponent)

% Score vs. Iteration

Single-Elimination Tournament

• Divide a population of n individuals to n/2 pairs and evaluate each pair
• The loser at each competition is assigned a fitness of 1/n, and the winner proceeds to the next round
• This process repeats itself until one individual remains with fitness 1

Single-Elimination Tournament

 Individuals
Benchmark (Coevolution)

Sample GP-Gammon

Comparison of Backgammon Players

Comparison of Backgammon Players

% Wins vs. Pubeval
Added Horsepower ...

- ... And went from 56% to 62%
- What about humans?
- Indirectly:
  - GP-Gammon: 62% vs. Pubeval
  - HC-Gammon: 40% vs. Pubeval
  - HC-Gammon against the (human) world:
    58% wins (counting abandoned as wins)
    38% wins (otherwise)
    (demo.cs.brandeis.edu/hcg/stats1.html)
- By transitivity ...

Comparing External Opponent with Coevolution

- Sole difference: Fitness measure
- We expected that individuals evolved by referring to external opponent would perform better against the same opponent, post-evolutionarily

Compare Approaches: Average

Compare Approaches: Max
Coevolution is Better (for backgammon)

- When playing only against one strategy, individuals are likely to adapt to the external opponent’s strategy.
- In order to overpower the external opponent, the individuals need motivation to discover new, unknown strategies.
- Hence, coevolution,
- or, as in GP-Robocode, use multiple externals.

So, Coevolution or External Opponent?

- Subtle
- Depends on
- One thing to evolve

Observation

- Studying the GP-Gammon individuals, we concluded that strategy is mostly due to general query functions.
- Specific query function helps "balance" strategy at critical moments.

The evolution of authority
The Game of Chess

- First developed in India and Persia
- A complex game of strategy and ingenuity
- Enormous search space: Estimated at $10^{43}$ in 40-move game (Shannon, 1950)

AI & Chess

- 1958: First chess AI program, novice level
- Over ~50 years, hardware gets really better
- Software...
- Well... Also
- 1997: Garry Kasparov, former world champion, defeated by IBM's Deep Blue
- So...

- NO!
- Deep Blue used extreme brute force, traversing several millions board positions
- Very little generalization
- Virtually no resemblance to human chess thinking
- Deemed theoretically uninteresting
**Chess Basics**

- 8x8 board
- Each player starts with 16 pieces of 6 different types, and may only move 1 piece per turn
- A piece can only move into an empty square or into one containing an opponent's piece (capture)
- Win by capturing the opponent's king

**Chess Moves**

- Pawn: May only move forward (or capture diagonally)
- Bishop: Diagonals
- Knight: L-shaped moves
- Rook: Ranks & files
- Queen: Bishop & Rook combined
- King: One square in any direction, may not move into attacked square

**Example**

- White has over 30 possible moves
- If black's turn, can capture pawn at c3 and check (also fork)

**Check and Checkmate**

- Checking: Attacking opponent's king
- Opponent must respond
- Mating: When the opponent runs out of move options, thus losing game
Artificial Players

• AI uses powerful search
• Millions of boards (search-tree nodes) per second
• Little time per board, less knowledge
• Smart search algorithms

Human Players

• Humans use (problem-solving) cognition
• Highly knowledge-based, extensive chess “theory”
• Massive use of pattern recognition
• Also use search but
  - Less deep
  - Only develop “good” positions
• More efficient, less nodes per “same” result
• Of course, said cognition used not only in chess...

(Human) Grand Masters

• Can play (well) against several opponents simultaneously
• GMs vs. novices: same level of performance when memorizing random board, differ when memorizing real game positions
• GM eye movements show they only scan “correct” parts of board
• Strong amateurs use same meta-search as GMs: Equally deep, same nodes, same speed; differ in knowledge of domain (De Groot)

Endgame: Example

• White’s turn: Mate in 5, with Qe6+
• Black’s turn: Draw with: Rc1+, then Qg5 - fork & exchange
Endgames

- Few pieces remain (typically: king, 0-3 officers and sometimes pawns)
- Fewer options, but more possible moves per piece
- Trees still extremely large

GP Building Blocks

- Main purpose: Reduce search by employing “smart” features of the board
- Allow more complex features to be built automatically by supplying basic ones (terminals) and building methods (functions)

Example Feature: Fork

- My piece:
  - Attacking 2 or more pieces
  - Protected or not attacked
- Opponent’s pieces:
  - unprotected
  - or, protected but of greater value
- Right: black must exchange Q for R

Fork: Traditional AI Search

- Black: 3 legal moves
- Find that one of white’s next moves (of 23) captures black queen
- Check all following moves for more piece exchanges
- Sometimes, still check other moves (non capturing)
- At end of search, compare remaining pieces
Fork in GP

- isMyFork function: checks all previously defined conditions
- Also, smaller building blocks:
  - Is opponent piece Attacked?
  - Is attacking piece protected?
  - Is opponent in check?
  - Value of attacked piece

Basic Program Architecture

- Generate all possible moves (depth=1)
- Evaluate each board with GP individual
- Select board with best score (or choose randomly between equals)
- Perform best move
- Repeat process with GP opponent until game ends (or until only kings left)
- 3 trees per individual: advantage, even, disadvantage, used according to current board status

Genome Summary

<table>
<thead>
<tr>
<th>Simple Terminals</th>
<th>Complex Terminals</th>
<th>Function Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsMyKingInCheck</td>
<td>EvaluateMaterial</td>
<td>IF(0,F,F)</td>
</tr>
<tr>
<td>IsOppKingInCheck</td>
<td>IsMaterialIncrease</td>
<td>Or(0,0,0)</td>
</tr>
<tr>
<td>MyKingDistEdges</td>
<td>IsMove</td>
<td>Or(0,0,0,0)</td>
</tr>
<tr>
<td>OppKingProximityToEdges</td>
<td>IsMateInOne</td>
<td>And(0,0)</td>
</tr>
<tr>
<td>NumMyPiecesNotAttacked</td>
<td>OppPieceCanBeCaptured</td>
<td>And(0,0,0)</td>
</tr>
<tr>
<td>NumOppPiecesAttacked</td>
<td>MyPieceCannotBeCaptured</td>
<td>Smaller(0,0,0)</td>
</tr>
<tr>
<td>ValueMyPiecesAttacking</td>
<td>IsOppKingStuck</td>
<td>Net(0)</td>
</tr>
<tr>
<td>ValueOppPiecesAttacking</td>
<td>IsMyKingNotStuck</td>
<td></td>
</tr>
<tr>
<td>IsMyQueenNotAttacked</td>
<td>IsOppKingBehindPiece</td>
<td></td>
</tr>
<tr>
<td>IsOppQueenNotAttacked</td>
<td>IsMyKingNotBehindPiece</td>
<td></td>
</tr>
<tr>
<td>IsMyFork</td>
<td>IsOppPiecePinned</td>
<td></td>
</tr>
<tr>
<td>IsOppFork</td>
<td>IsMyPieceNotPinned</td>
<td></td>
</tr>
<tr>
<td>NumMovesMyKing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumNotMovesOppKing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyKingProxBook</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OppKingDistRank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyPiecesSameLine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OppPiecesNotSameLine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsOppKingProtecting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsMyKingProtecting</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two (Top) Opponents

Evolved programs tested against two top-rated players:

1. Program we wrote based on consultation with experts, highest being International Master Boris Gutkin, ELO 2400
2. CRAFTY, second in the 2004 International Computer Chess Championship

Endgame Experiments

- KRKR: Each player has 1 king and 1 rook
- KRRKR: King with 2 rooks vs. king and rook
- KRRKRR
- KQKQ: Kings and Queens
- KQRKQR: Combined
Sample GP-Endchess

Tree 0:
(If3 (Or2 (Not (And2 OppPieceAttUnprotected NotMyKingInCheck)) (Or2 NotMyPieceAttUnprotected 100*Increase))
(And2 (Or3 (And2 OppKingStuck NotMyPieceAttUnprotected) (And2 OppPieceAttUnprotected OppKingStack) (And3 -
1000*MateInOne OppKingInCheckPieceBehind NotMyKingStuck)) (Or2 (Not NotMyKingStuck) OppKingInCheck))))
NumMyPiecesUNATT (If3 (If3 (Or2 NotMyPieceAttUnprotected NotMyKingInCheck) (If3 NotMyPieceAttUnprotected #NoMovesOppKing OppKingInCheckPieceBehind) (If3 OppKingStack OppKingInCheckPieceBehind -1000*MateInOne))
(If3 (And2 100*Increase 1000*MateInOne)) (If3 (NumMyPiecesUNATT (If3 NotMyPieceAttUnprotected -1000*MateInOne)
OppKingInCheckPieceBehind)) (If3 (MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne MatchInOne -1000*MateInOne
NotMyPieceATT)) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind) NumOppPiecesATT) (If3 (And3 -
1000*MateInOne NotMyPieceAttUnprotected 100*Increase)) (If3 (NumMyPiecesUNATT (If3 NotMyPieceAttUnprotected -
1000*MateInOne OppKingProxEdges)) (If3 (MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne MatchInOne -
1000*MateInOne NotMyPieceATT)) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind) NumOppPiecesATT)
(If3 (And3 -1000*MateInOne NotMyPieceAttUnprotected 100*Increase)) (If3 (NumMyPiecesUNATT (If3 NotMyPieceAttUnprotected -
1000*MateInOne OppKingProxEdges)) (If3 (MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne MatchInOne -
1000*MateInOne NotMyPieceATT)) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind) NumOppPiecesATT)
(If3 NotMyPieceATT (If3 NotMyPieceAttUnprotected #NoMovesOppKing) (If3 1000*Mate?) Tree 1:
(If3 NotMyPieceATT (If3 NotMyPieceAttUnprotected #NoMovesOppKing) (If3 1000*Mate?)
((If3 NotMyPieceATT (If3 NotMyPieceAttUnprotected #NoMovesOppKing) (If3 1000*Mate?)
(If3 OppPieceAttUnprotected NumMyPiecesUNATT MyFork)))))

Tree 1:
(If3 NotMyPieceAttUnprotected #NoMovesOppKing 1000*Mate?)

Tree 2:
(If3 1000*Mate? NumMyPiecesUNATT -1000*MateInOne)

Multiple Endgames

• Aim for general-purpose strategies
• All endgames used during evolution
• Results:

<table>
<thead>
<tr>
<th></th>
<th>%Wins</th>
<th>%Adv</th>
<th>%Draws</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>6</td>
<td>2</td>
<td>68</td>
</tr>
<tr>
<td>CRAFTY</td>
<td>2</td>
<td>4</td>
<td>72</td>
</tr>
</tbody>
</table>

Evolution of an Efficient Search Algorithm for the Mate-In-N Problem in Chess

Ami Hauptman and Moshe Sipper
Ben-Gurion University, Israel

2007 "HUMIES" AWARDS FOR HUMAN-COMPETITIVE RESULTS

Monday, July 9, 2007

Game-Playing AI

• Game Strategy = Search + Knowledge
• Search:
  • Number of nodes developed
• Knowledge:
  • Evaluation of nodes
• Tradeoff between the two
**Chess: Machine Players**

- Powerful contemporary engines
  - *Crafty*, Fritz, Deep Junior, ... 
  - Lots of search 
  - Less knowledge
- Intelligent? Hmmm...
  - Very little generalization 
  - Gobbles computational power 
  - Deemed theoretically uninteresting [Chomsky, 93]

**Our Goal**

- Concentrating on endgames we previously: 
  - evolved node-evaluation function (knowledge) with GP 
  - Results: draw or win against CRAFTY, a world-class chess engine  
  - Part of work that won a 2005 humies medal
- This work: Evolve the search algorithm itself
  - Evolve both search and knowledge, letting evolution balance the two

**Chess: Human Players**

- Use problem solving cognition
- Deeply knowledge-based play
- Massive use of pattern recognition; parallelism 
- Also use search but 
  - Substantially less nodes (typically dozens) 
  - Selective (only "good")
  - More efficient: less nodes for “same” result
- Good source of inspiration for algorithms

**Incentive for Current Work**

- Previously evolved players: 
  - Sometimes miss (easy) shallow mates 
  - Scaling problem: adding pieces to board decreased scores
- Evolved players should rely more on search
  - Full pure-knowledge player still unattainable 
  - Search makes the strongest engines
- Problem:
  - Simply adding search: too slow (each node thoroughly examined)

⇒ SOLUTION: 
  - Balancing search & knowledge through evolution
**Problem Domain**

- Mate-in-N: Is there a forced win sequence in maximum $2^*(N-1)$ plies?
- Crucial to chess engines, searched far more thoroughly.
- CRAFTY: For difficult $N=5$ cases searches over $10^6$ nodes.

<table>
<thead>
<tr>
<th>Mate-in</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth in plies</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Nodes developed</td>
<td>600</td>
<td>7K</td>
<td>50K</td>
<td>138K</td>
<td>1.6M</td>
</tr>
</tbody>
</table>

**Result is Human-Competitive**

(H) result holds its own or wins a regulated competition involving human-written computer programs.
(B) better than result accepted as a new scientific result at the time.
(D) result is publishable in its own right.
(F) better than result considered an achievement at the time.
(G) result solves a problem of indisputable difficulty in its field.

**Major Result**

Evolved search algorithm:

Number of nodes developed reduced by 47% with respect to world-class engine (not simple $\alpha\beta$).

<table>
<thead>
<tr>
<th>Mate-in</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAFTY</td>
<td>600</td>
<td>7K</td>
<td>50K</td>
<td>138K</td>
<td>1.6M</td>
</tr>
<tr>
<td>Evolved</td>
<td>600</td>
<td>2k</td>
<td>28k</td>
<td>55K</td>
<td>850k</td>
</tr>
</tbody>
</table>

**Why is Result Best?**

- Difficult for most human chess players:
  - Must train intensively not to miss (and lose game).
- Our evolved strategies improve upon one of top chess engines in existence (Crafty), representing many human years of programming.
- We're beating this top-notch engine in its own "territory": massive search.
- Problem is crucial to chess engines, therefore much computational power is expended (e.g., in such positions, Deep Blue examines twice the normal number of nodes).
Why is Result Best? (cont’d)

- Evolving a dynamic algorithm (i.e., a process) usually harder than evolving a static structure
- We took evolution to the next level: balancing search and knowledge
- Surpasses previous EC solutions

In a nutshell:

1. Hard problem in hard domain for man & machine (chess)
2. Evolved algorithm better than (most) humans
3. Evolved algorithm better than human-written top engine
4. Evolution taken to next level

To Briefly Summarize...

- Genetic Programming has proven to be an excellent tool for automatically creating game strategies
- Robocode: 2nd place in international league, with other 26 programs written by humans
- Backgammon: highest win rate vs. Pubeval, most likely will hold its own against humans
- Chess: draw/win vs. top-rated, human-based programs

General Discussion
Automatic Programming

Koza et al. (1999) delineated 16 attributes a system for automatically creating computer programs might beneficially possess:

1. Starts with problem requirements
2. Produces tractable and viable solution to problem
3. Produces an executable computer program
4. Automatic determination of program size
5. Code reuse
6. Parameterized reuse
7. Internal storage
8. Iterations, loops, and recursions
9. The ability to organize chunks of code into hierarchies
10. Automatic determination of program architecture
11. Ability to implement a wide range of constructs known to programmers
12. Operates in a well-defined manner
13. Problem-independent, i.e., possesses some degree of generalization capabilities
14. Applicable to a wide variety of problems from different domains
15. Able to scale well to larger instances of a given problem
16. Competitive with human-produced results

Attribute 17

Cooperative with Humans

• GP readily accommodates human expertise
• Common AI view: Yada from Nada
• Like Athena, Greek goddess of wisdom, favorite child of Zeus, who had sprung fully grown out of her father's head
• Cooperative view: Man and machine work hand-in-keyboard
Cooperative with Humans

- More than many (most?) other adaptive-search technicians, the GPer is better positioned to imbue the machine with his own intelligence

\[ \text{Genetic Programming + (human) Intelligence} \rightarrow \text{Human-Competitiveness} \]

A/I or A-I?

- Importance of high “artificial-to-intelligence ratio” (Koza et al., 2003): Maximize A/I.
- Better: \( A - I \geq M_c \) (meaningful epsilon)
- Pore in as much I as possible, with goal of maximizing (machine’s) added Value: A

The Ultimate Goal...

Building a Strategizing Machine

Warning: The Surgeon General Has Determined That The Following May Be Totally Nonsensical

A Final Thought...

Machines Might Be Human-Competitive
BUT...

Are Humans Machine-Competitive?

Human Cellular Automata
[parallel computer made of simple, 2-state processors]
Human Cellular Automata

- Slow
- Error-prone
- Processors tend to complain

Speaking of (brilliant) students...

- yaniv azaria
- ami hauptman
- yehonatan shichel
- eran ziserman

In conclusion...

The manifestation of the universe as a complex idea unto itself as opposed to being in or outside the true Being of itself is inherently a conceptual nothingness or Nothingness in relation to any abstract form of existing or to exist or having existed in perpetuity and not subject to laws of physicality or motion or ideas relating to non-matter or the lack of objective Being or subjective otherness.

Woody Allen, Mr. Big
www.moshesipper.com/papers

- Y. Azaria and M. Sipper
  GP-GAMMON: USING GENETIC PROGRAMMING TO EVOLVE BACKGAMMON PLAYERS
  Proceedings EuroGP2005, pp. 132-142

- A. Hauptman and M. Sipper
  GP-ENDCHESS: USING GENETIC PROGRAMMING TO EVOLVE CHESS ENDGAME PLAYERS
  Proceedings EuroGP2005, pp. 120-131

- Y. Shichel, E. Ziserman, and M. Sipper
  GP-ROBOCODE: USING GENETIC PROGRAMMING TO EVOLVE ROBOCODE PLAYERS
  Proceedings EuroGP2005, pp. 143-154

- Y. Azaria and M. Sipper
  GP-GAMMON: GENETICALLY PROGRAMMING BACKGAMMON PLAYERS
  Genetic Programming and Evolvable Machines, vol. 6, no. 3, pp. 283-300, 2005

- M. Sipper, Y. Azaria, A. Hauptman, & Y. Shichel
  DESIGNING AN EVOLUTIONARY STRATEGIZING MACHINE FOR GAME PLAYING AND BEYOND

- A. Hauptman and M. Sipper
  EVOLUTION OF AN EFFICIENT SEARCH ALGORITHM FOR THE MATE-IN-N PROBLEM IN CHESS
  Proceedings EuroGP2007, pp. 78-89