Evolutionary Computation & Games

ben-gurion university, israel
www.moshesipper.com

GECCO'08, July 12-16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-131-6/08/07.

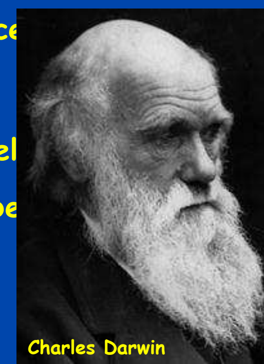
Outline

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

2

Genetic Algorithms (GA)

- A class of probabilistic optimization algorithms
- Inspired by biological proce y
  Natural Selection
- Uses analogs of natural sel
  operators (cf. guy-with-be


Charles Darwin

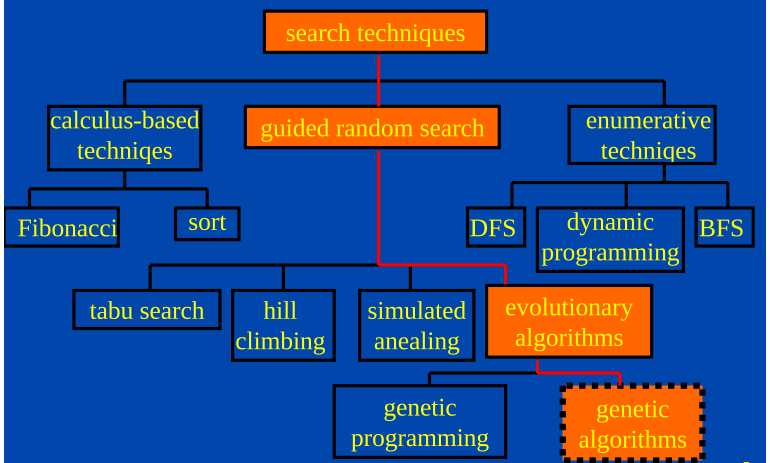
3

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

4

Classes of Search Techniques

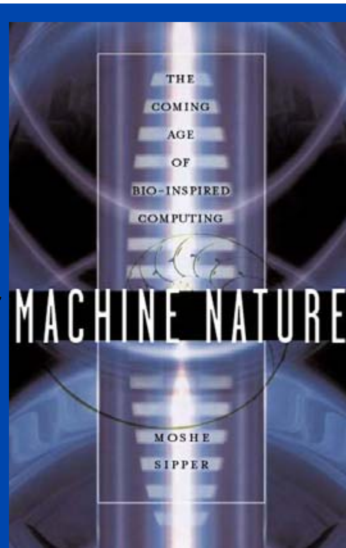


5

genetic algorithms are a form of bio-inspired computing

speaking of which…

Machine Nature: The Coming Age of Bio-Inspired Computing
McGraw-Hill, New York, 2002



a genetic algorithm maintains a population of candidate solutions for the problem at hand, and evolves this population by iteratively applying a set of stochastic operators, driven by fitness

7

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

8

Simple Genetic Algorithm

produce initial population of individuals

evaluate fitness of all individuals

while *termination-condition* not met do

  select fitter individuals for reproduction

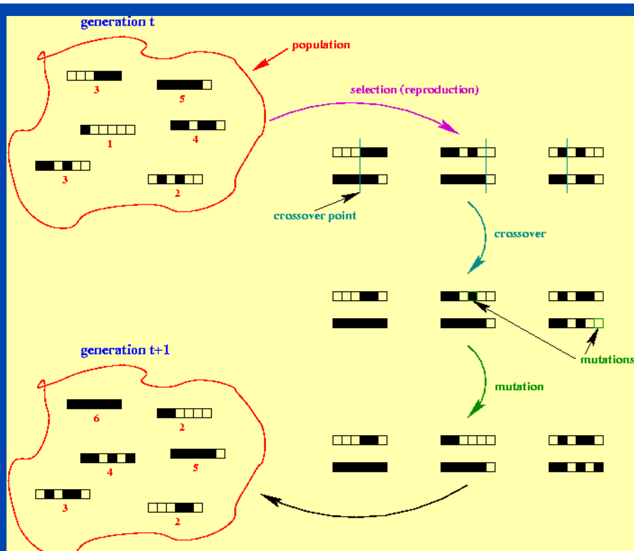
  recombine individuals (crossover)

  mutate individuals

  evaluate fitness of modified individuals

end while

9



10

The Metaphor

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

11

artificial evolution is highly simplified relative to biology

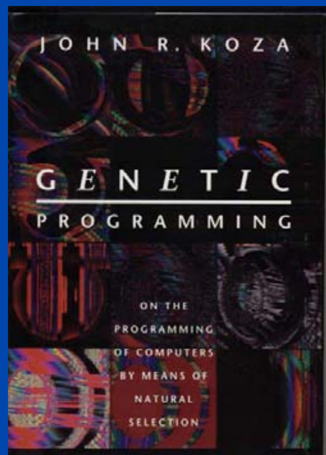
BUT

repeatedly produces surprisingly complex, interesting, and useful solutions

12

Genetic Programming (GP)

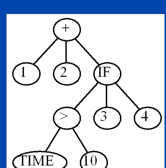
- Nichael Cramer, "A Representation for the Adaptive Generation of Simple Sequential Programs," 1985
- Transformed into an effervescent field in large part due to John Koza



13

Genetic Programming

- GP = GA, with individuals in population represented as computer programs
- Usually LISP programs (S-Expressions)
- Genome composed of *functions* and *terminals*
- GPer determines function set and terminal set

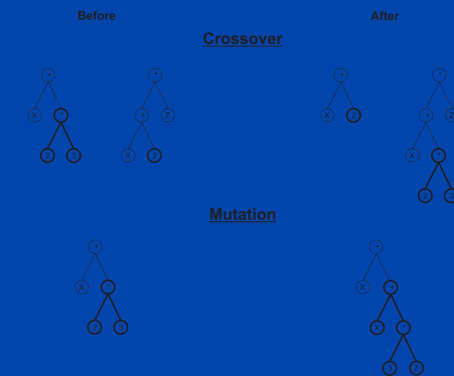



```
(+ 1 2 (IF (> TIME 10) 3 4))
```

14

Genetic Programming

- Why LISP? Easier (but not easy) to design genetic operators



15

Applying GP

To apply GP, specify:

1. Terminal set
2. Function set
3. Fitness measure
4. Control parameters (p_{cross}, p_{mut}, p_{rep}, …)
5. Termination criterion, result designation

16

Example: Symbolic Regression

(Koza)

Independent variable X	Dependent variable Y
-1.00	1.00
-0.80	0.84
-0.60	0.76
-0.40	0.76
-0.20	0.84
0.00	1.00
0.20	1.24
0.40	1.56
0.60	1.96
0.80	2.44
1.00	3.00

17

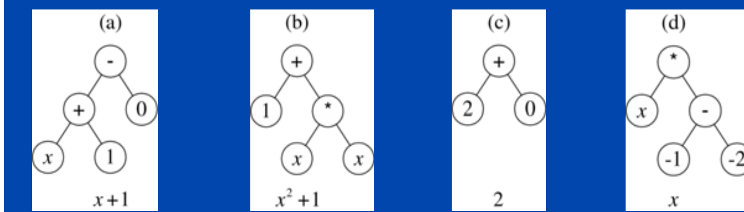
Setup

	Objective:	Find computer program with one input (X) whose output equals given data
1	Terminal set:	`T = {X, Random-Constants}`
2	Function set:	`F = {+, -, *, %}`
3	Fitness:	Σ abs (program output - given data)
4	Parameters:	Population size M = 4
5	Termination:	Individual emerges with absolute error < 0.1

18

Initial Population

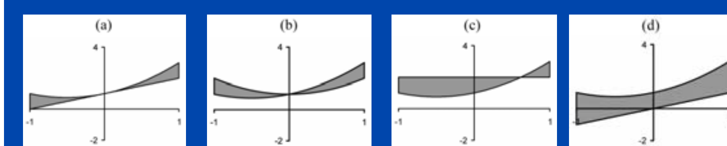
4 randomly created individuals of generation 0



19

Symbolic Regression $x^2 + x + 1$

fitness of the 4 individuals in generation 0

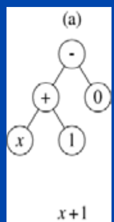


$x + 1$	$x^2 + 1$	2	x
0.67	1.00	1.70	2.67

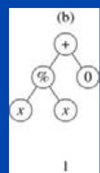
20

Symbolic Regression $x^2 + x + 1$

generation 1

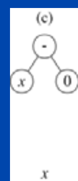


Copy of (a)

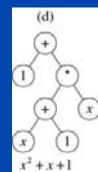


Mutant of (c)

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

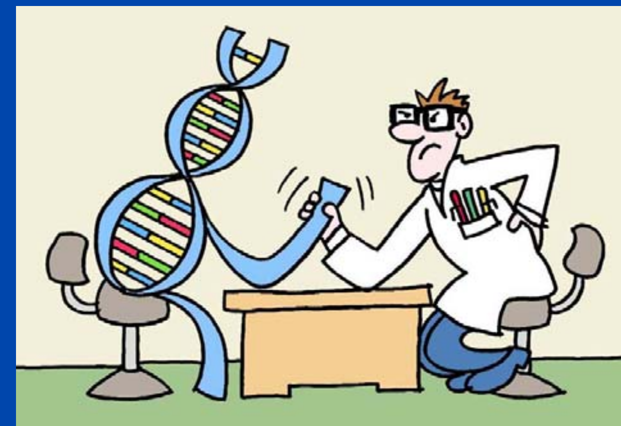
21

Genetic Programming

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

22

Human-Competitive Results



23

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)

24

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!

25

Games

26

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

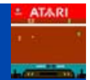
27

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)

28

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)

29

Robocode

30

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

31

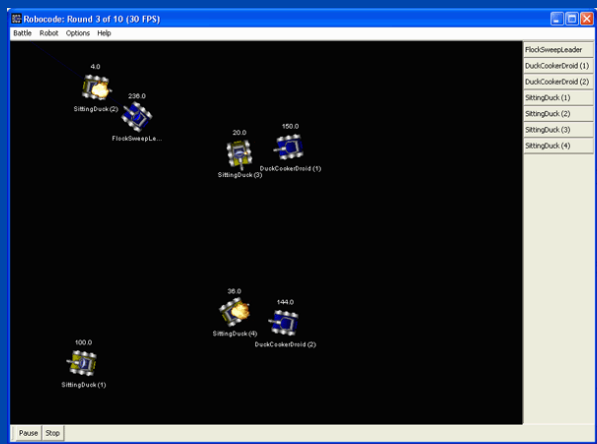
Robocode Player

Java program, Event driven

```
package sample;
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);
    }
```

32

Simulator Interface



33

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

34

Goal

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

35

Applying GP

- Tree genome implementing sub-programs
- Main
- Handling of specific events
  - onScannedRobot
  - onHitWall
  - onHitRobot
- Main functions
  - Move
  - TurnTank
  - TurnGun
  - TurnRadar
  - Fire

36

Robocode Player's Code Layout

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

37

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

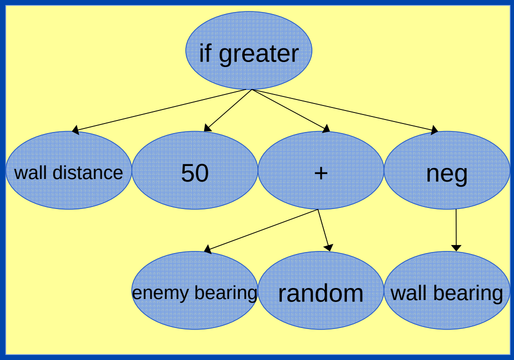
38

Genome Summary

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

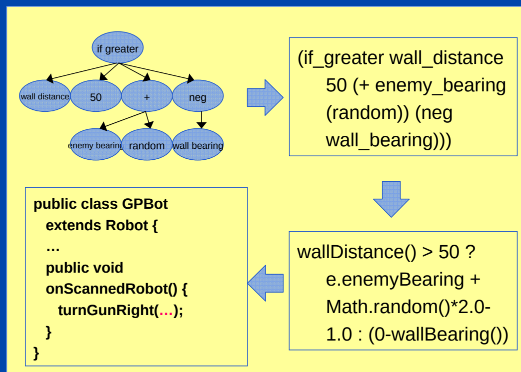
39

Sample Program



40

Genotype to Phenotype



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):

$$F = \frac{\epsilon + Sp}{\epsilon + S_P + S_A}$$

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

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)

Sample GP-Robocode

```
package geep.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)          {
                    setAhead((getEnergy() > Math.abs(Math.abs(getEnergy()) )  ? getEnergy() : (getHeadingRadians() > getY() ? getEnergy() :
Math.sin(getEnergy())) + Math.abs(e.getHeadingRadians()) ) + ((setFireBullet(Math.abs(Math.abs(e.getEnergy()) ) )==null ? 0.0 : 1.0) > 0 ?
e.getEnergy() : Math.abs(Math.abs(getBattleFieldWidth()) )  + Math.abs(Math.sin(getEnergy())) ) + Math.abs(((0-(((getHeadingRadians() >
getY() ? getY() : e.getBearingRadians()) > 0 ? e.getEnergy() : getEnergy()))) + Math.sin(getEnergy()) > 0 ? (getEnergy() > getY() ?
e.getBearingRadians() : e.getBearingRadians()) : (getEnergy() > 0 ? e.getEnergy() : getEnergy())))  + getBattleFieldWidth());
                    setTurnRightRadians(robocode.util.Utils.normalRelativeAngle((0-(Math.abs(Math.sin(Math.abs(getBattleFieldWidth())  +
getBattleFieldWidth() + Math.sin((Math.sin(getEnergy()) > 0 ? e.getEnergy() : getY()))) - (setFireBullet(Math.abs(((getY() > 0 ? e.getEnergy() :
getEnergy()) > getY() ? getEnergy() : e.getBearingRadians()))  + Math.abs((setFireBullet(((getBattleFieldWidth() > 0 ? e.getEnergy() :
e.getEnergy()) > 0 ? e.getEnergy() : getEnergy()))==null ? 0.0 : 1.0)) )==null ? 0.0 : 1.0)) )))));
                    setTurnGunRightRadians(robocode.util.Utils.normalRelativeAngle(getHeadingRadians()-
getGunHeadingRadians()+(getHeadingRadians() > getY() ? Math.abs(Math.sin(Math.sin((getHeadingRadians() > getY() ? getEnergy() :
Math.abs(getHeadingRadians()) )) + Math.abs((((getHeadingRadians() > getY() ? getEnergy() : e.getBearingRadians()) > 0 ? e.getEnergy() :
Math.abs(getBattleFieldWidth()) ) > (e.getBearingRadians() > 0 ? e.getEnergy() : getY()) ? getEnergy() : (getEnergy() > 0 ? e.getEnergy() :
getEnergy()))) ) - (setFireBullet(Math.sin((setFireBullet((Math.abs(e.getEnergy())  > 0 ? e.getEnergy() : getEnergy()))==null ? 0.0 : 1.0)) +
Math.abs((Math.abs(getHeadingRadians())  > (getHeadingRadians() > getY() ? getEnergy() : (getHeadingRadians() > getY() ? getEnergy() :
e.getBearingRadians())) ? getEnergy() : e.getBearingRadians())) )==null ? 0.0 : 1.0))  : e.getBearingRadians())));          }}/*  > Tree 0  ) add (add
(add (ifGreater energy (abs (abs energy)) energy (add (ifGreater heading y energy (sin energy)) (abs enemy_heading))) (ifPositive (fire (abs (abs
enemy_energy))) enemy_energy (add (abs (abs max_x)) (abs (sin energy))))) (abs (ifPositive (add (neg (ifPositive (ifGreater heading y y
enemy_bearing) enemy_energy energy)) (sin energy)) (ifGreater energy y enemy_bearing enemy_bearing) (ifPositive energy enemy_energy
energy)))) max_x)  > Tree 1   ) neg (abs (sub (sin (add (abs max_x) (add max_x (sin (ifPositive (sin energy) enemy_energy y))))) (fire (add (abs
(ifGreater (ifPositive y enemy_energy energy) y energy enemy_bearing)) (abs (fire (ifPositive (ifPositive max_x enemy_energy enemy_energy)
enemy_energy energy))))))))  > Tree 2          ) ifGreater heading y (abs (sub (sin (add (sin (ifGreater heading y energy (abs heading))) (abs
(ifGreater (ifPositive (ifGreater heading y energy enemy_bearing) enemy_energy (abs max_x)) (ifPositive enemy_bearing enemy_energy y)
energy (ifPositive energy enemy_energy energy))))) (fire (add (sin (fire (ifPositive (abs enemy_energy) enemy_energy energy))) (abs (ifGreater
(abs heading) (ifGreater heading y energy (ifGreater heading y energy enemy_bearing)) energy enemy_bearing)))))) enemy_bearing/*(
```

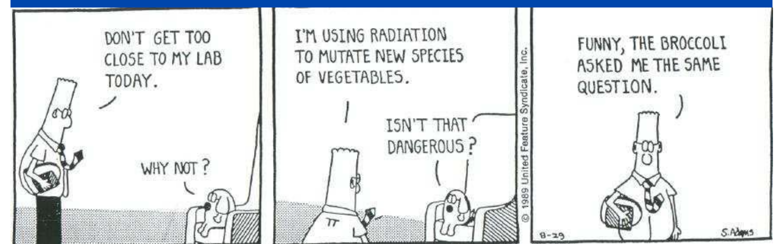
45

GP-Robocode

Rank	Rating	Robot	Total Score	Survival	Last surv	Bullet dmg	Bonus	Ram dmg	Bonus	1sts	2nds	3rds
1	185.29	geep.haiku.GPBotC 1.0	32179	9450	1890	16012	2363	2077	377	197	125	0
2	168.79	kawigi.haiku.HaikuTrogdor 1.1	37822	12650	2530	19080	3292	233	29	255	67	0
3	141.67	cx.haiku.MeleeXaxa 1.0	32593	11000	2200	16022	2181	857	324	223	98	0
4	140.57	pez.femto.HaikuPoet 0.2	26241	9750	1950	12083	1862	569	21	202	121	0
5	126.76	kawigi.femto.FemtoTrogdor 1.0	31527	8800	1760	15337	2022	3138	462	187	[illegible]	0
6	120.27	ms.AresHaiku 0.4	29067	9050	1810	15881	2177	143	[illegible]	[illegible]	[illegible]	0
7	118.11	cr.OneOnOneHaiku 1.1	41943	9600	1920	22645	3185	[illegible]	[illegible]	[illegible]	[illegible]	0
8	105.53	mz.HaikuGod 1.01	31604	11800	2360	12986	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
9	67.20	kawigi.haiku.HaikuChicken 1.0	24581	6900	1380	[illegible]	[illegible]	[illegible]	[illegible]	189	131	0
10	61.81	pez.femto.WallsPoetHaiku 0.1	25739	7950	1590	[illegible]	[illegible]	[illegible]	168	163	161	0
11	60.79	kawigi.haiku.HaikuCircleBot 1.0	32831	10900	[illegible]	[illegible]	[illegible]	344	89	222	98	0
12	36.17	soup.haiku.RammerHK 1.0	41258	[illegible]	[illegible]	[illegible]	2260	8219	999	150	171	0
13	27.97	kawigi.haiku.HaikuSillyBot 1.2	[illegible]	[illegible]	[illegible]	13524	1461	429	182	166	155	0
14	20.13	kawigi.haiku.HaikuLinearAimer 1.0	[illegible]	[illegible]	1570	17367	2349	273	54	164	[illegible]	0
15	11.13	cx.haiku.Escape 1.0	[illegible]	[illegible]	2180	10989	1856	177	0	222	[illegible]	0
16	7.40	cx.haiku.Xaxa 1.1	[illegible]	11350	2270	17694	2967	180	14	230	[illegible]	0
17	-14.08	[illegible]	19705	8250	1650	8650	1051	72	25	181	[illegible]	0
18	-24.32	[illegible]	21011	3950	790	14378	1347	513	[illegible]	141	[illegible]	0
19	[illegible]	[illegible]ango 1.0	18769	4700	940	11631	1089	375	[illegible]	[illegible]	[illegible]	0
20	[illegible]	[illegible]DodgeHK 1.0	22354	7850	1570	11374	941	563	[illegible]	[illegible]	[illegible]	0
21	[illegible]	[illegible].ChaosHaiku 0.1	28704	7850	1570	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
22	[illegible]	cx.haiku.Smoku 1.1	22928	5300	1060	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
23	-87.01	klo.haikuBounC 1.0	21675	6000	1200	12952	1260	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
24	-87.36	soup.haiku.RandomHK 1.0	19712	4500	900	12982	1156	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
25	-136.47	soup.haiku.CutoffHK 1.0	26332	4250	850	13519	1247	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
26	-177.37	davidalves.net.PhoenixHaiku 1.0	19896	5450	1090	12196	1018	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
27	-208.72	dummy.haiku.Disoriented 1.0	17946	4100	820	12171	775	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
28	-478.37	soup.haiku.WallDroidHK 1.0	6271	2050	410	3301	105	329	68	46	[illegible]	0

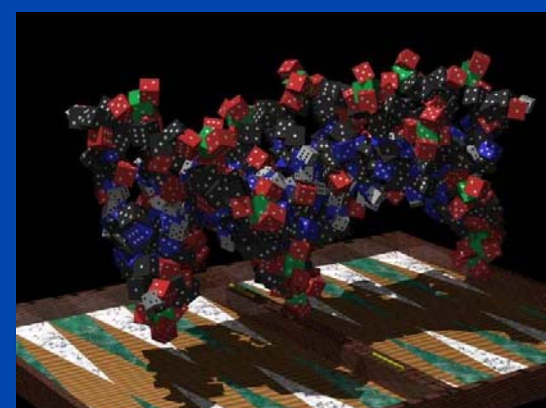
All Other 27 Players: Human Written!

June 25, 2005 (http://robocode.yajags.com/20050625/haiku-1v1.html) 46



47

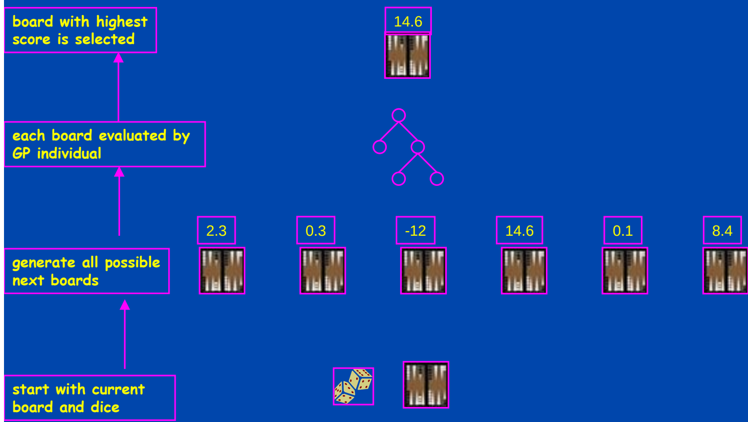
Backgammon



48

Evolving Backgammon Players with GP

Goal: Evolve a good flat evaluator



Something About Backgammon

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

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

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: atomic = symbol, set of atomic types
- Need to define type constraints for backgammon:
  - atomic : *Float* (F), *Boolean* (B)
  - set: *Query* (Q), 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

53

Constants

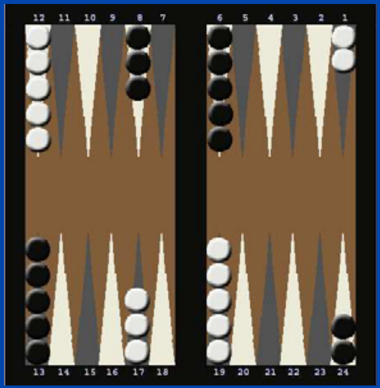
- An ephemeral random constant (ERC) produces a constant real number in the range [0,5]:

$$F = \mathit{Float\text{-}ERC}$$

54

Specific Queries

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



55

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:
  - *Player-Exposed(n)*
  - *Player-Blocked(n)*
  - *Player-Tower(n)*
  - *Enemy-Exposed(n)*
  - *Enemy-Blocked(n)*

$$Q = \mathit{Specific\text{-}Property}(n)$$

56

General Board Queries

- Provide general information about the board configuration
- General properties include:

Player-Pip

Enemy-Pip

Total-Hit-Prob

Player-Escape

Enemy-Escape

$$Q = \textit{General-Property}$$

57

Terminal Set for Race Tree

- Much simpler and contains the functions:

$$F = \textit{Float-ERC}$$

$$Q = \textit{Player-Position}(n)$$

58

Function Set

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

59

Function Set (cont'd)

- Arithmetic functions:

  F=Add(F,F) F=Sub(F,F) F=Mul(F,F)
- Conditional functions:

  F=If(B,F,F)
- Compare functions:

  B≥(F,F) B≤(F,F)
- Logic function:

  B=And(B,B) B=Or(B,B) B=Not(B)

60

Genome Summary

Terminal Set: Contact Tree	Terminal Set: Race Tree	Function Set: Both Trees
F=Float-ERC	F=Float-ERC	F=Add(F,F)
Q=Player-Exposed(n)	Q=Player-Position(n)	F=Sub(F,F)
Q=Player-Blocked(n)		F=Mul(F,F)
Q=Player-Tower(n)		F=If(B,F,F)
Q=Enemy-Exposed(n)		B=Greater(F,F)
Q=Enemy-Blocked(n)		B=Smaller(F,F)
F=Player-Pip		B=And(B,B)
F=Enemy-Pip		B=Or(B,B)
F=Total-Hit-Prob		B=Not(B)
F=Player-Escape		
F=Enemy-Escape		

61

First Approach: External Opponent

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

62

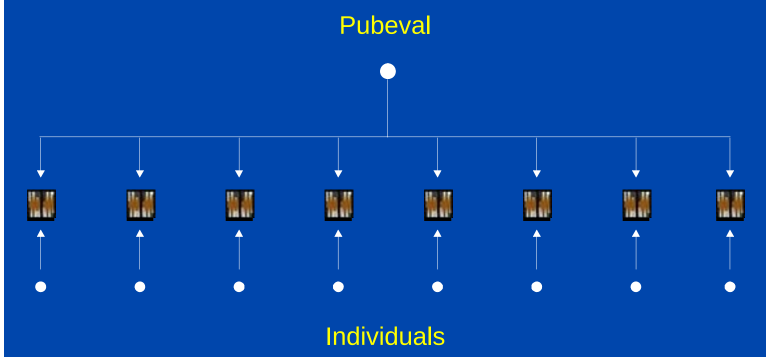
Fitness Measure

- Uses external opponent to evaluate individuals
- Each individual plays a 100-game tournament vs. Pubeval
- Fitness of individual (i=individual, p=pubeval):

$$F = \frac{Si}{Si + Sp}$$

63

Fitness Measure



64

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

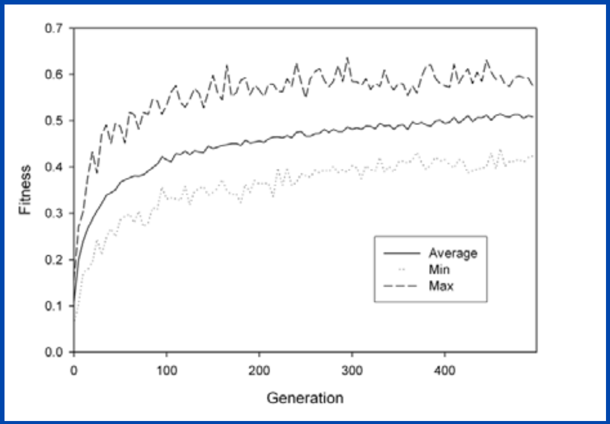
65

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

66

Results (External Opponent)



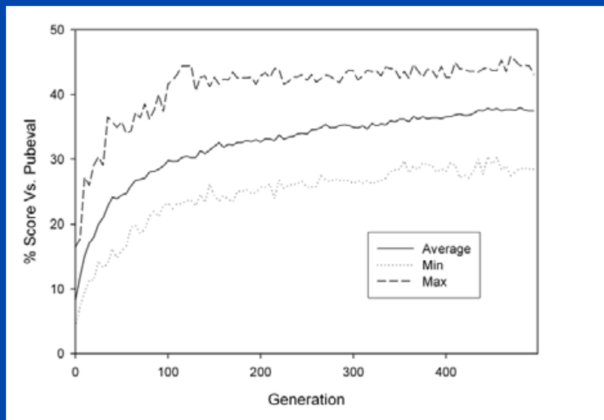
67

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

68

Benchmark (External Opponent)



Coevolution

- Results obtained using external opponent approach are good — but we wanted

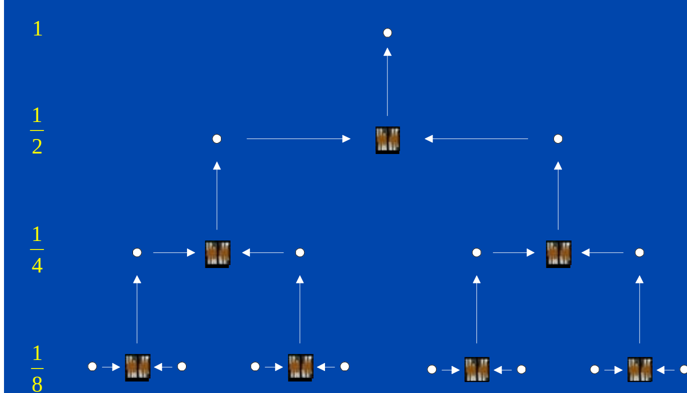
  Better. Stronger. Faster.

- Suspected to be over-fitted
- Apply coevolution: Individuals play against each other
- Use Single-Elimination Tournament (Angeline *et al.* 1993)

Single-Elimination Tournament

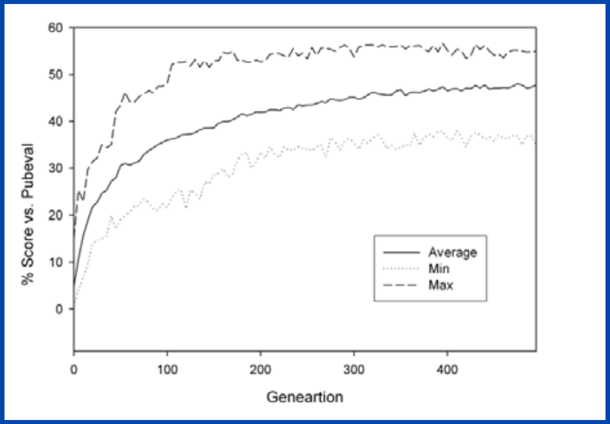
- Divide a population of n individuals to n/2 pairs and evaluate each pair
- The looser at each competition is assigned a fitness of 1/n, and the winner proceeds to the next round
- This process repeats itself until one individual remains with fitness 1

Single-Elimination Tournament



Individuals

Benchmark (Coevolution)



73

Sample GP-Gammon

```
Tree 0:
(+ (- (+ (+ (- (E-At 24) (- (+ (* (+ (If (P-Block 5) (P-Open 24) L-Pip) (+ (- (E-At 18) L-Pip) (+ EnemyEsc (+ E-Pip (P-Open 1))))) (P-Block 2)) (If (Not (P-Block 8)) (+ L-Pip EnemyEsc) (+ (- (E-At 18) L-Pip) (+ EnemyEsc (P-Block 3))))) (- (+ L-Pip (* 0.053718492 (E-At 11))) (If (Not (P-Block 8)) (+ (P-Block 1) TotalHit) (+ (+ E-Pip (P-Open 1)) (+ EnemyEsc (- (+ (* (+ L-Pip EnemyEsc) PlayerEsc) (+ EnemyEsc EnemyEsc)) (- (+ E-Pip (P-Open 1)) (If (Not (P-Block 8)) (+ E-Pip (P-Open 1)) (- P-Pip (P-Open 1))))))))))) (- (- 0.3715206 (* L-Pip (* EnemyEsc (P-Block 10)))) (* (- 2.9071698 L-Pip) (+ L-Pip (+ L-Pip (+ L-Pip EnemyEsc)))))) (- (- 0.3718665 (+ (P-Block 1) TotalHit)) (* (- 2.9086428 L-Pip) (+ L-Pip (+ L-Pip (If (< (If (P-Tower 5) PlayerEsc L-Pip) (If (Not (P-Block 8)) (If (Not (P-Block 8)) (+ (P-Block 1) (+ PlayerEsc TotalHit)) (+ EnemyEsc (+ EnemyEsc (P-Block 3)))) (+ (- (E-At 18) L-Pip) (- 2.9086428 L-Pip)))) (+ (If (P-Block 5) (* 1.1556402 (P-Block 2)) L-Pip) (* 1.1556402 (P-Block 2))) (If (> (+ (* (+ L-Pip EnemyEsc) (If (P-Tower 5) PlayerEsc (If (P-Block 5) (P-Open 24) L-Pip))) (+ EnemyEsc EnemyEsc)) PlayerEsc) (P-Open 1) (* 0.053718492 (E-At 11))))))))) (- (+ (+ EnemyEsc (P-Block 3)) (If (P-Block 5) (P-Open 24) L-Pip)) (- (If (Not (And (P-Open 25) (And (P-Open 10) (E-At 12)))) L-Pip EnemyEsc) (If (Not (P-Block 8)) (+ (P-Block 1) (* (- 2.9086428 (* 0.053718492 (E-At 11))) (+ (+ L-Pip (* 0.053718492 (E-At 11))) (+ L-Pip (If (> PlayerEsc PlayerEsc) (- (+ E-Pip (P-Open 1)) (If (Not (P-Block 8)) (+ E-Pip (P-Open 1)) (* 0.053718492 (E-At 11)))) (If (> PlayerEsc PlayerEsc) (+ L-Pip (+ L-Pip (If (< (* 1.1556402 (P-Block 2)) (- P-Pip (P-Open 1))) (+ (If (P-Block 5) (P-Open 24) L-Pip) (* 1.1556402 (P-Block 2))) (If (> PlayerEsc PlayerEsc) (P-Open 1) (* 0.053718492 (E-At 11)))))) L-Pip)))))) (- (E-At 18) L-Pip))))) (- (* 0.044896092 (+ (- (+ L-Pip (+ EnemyEsc EnemyEsc)) (+ EnemyEsc (P-Block 3))) (+ L-Pip (If (Not (P-Block 8)) (+ L-Pip PlayerEsc) (If (> PlayerEsc PlayerEsc) (- (E-At 18) L-Pip) L-Pip))))) (* EnemyEsc (+ L-Pip (+ (* (- P-Pip (P-Open 1)) (+ L-Pip EnemyEsc)) (If (< L-Pip (+ (If (< (- 2.9086428 L-Pip) PlayerEsc) (+ (* 1.1556402 (P-Block 2)) (* 1.1556402 (P-Block 2))) (If (> PlayerEsc PlayerEsc) (+ L-Pip (+ EnemyEsc (P-Block 3))) (* 0.053718492 (E-At 11)))) P-Pip)) (+ (If (P-Block 5) (P-Open 24) L-Pip) (* 1.1556402 (P-Block 2))) (If (> PlayerEsc (+ L-Pip EnemyEsc)) (+ (* 0.053718492 (E-At 11))
Tree 1:
(- 1.0983202 (+ 3.5341604 (- (P-Position 11) (* (* (P-Position 25) (* (- 0.9336438 (P-Position 15)) (+ (* 2.0106103 (P-Position 2)) 0.180087))) (- (* (P-Position 9) (- (+ 3.4345493 (If (< (P-Position 6) 2.1664124) (P-Position 18) (If (P-Position 21) (P-Position 17) (* (- 0.55943674 (If (P-Position 11) (* 2.0106103 (P-Position 2)) (P-Position 22))) (If (> (P-Position 18) (P-Position 16)) (If (P-Position 20) (P-Position 21) (P-Position 10)) (* (P-Position 25) (P-Position 25))))))) (- (- 0.27137187 (P-Position 25)) (+ (If (P-Position 15) (P-Position 5) 1.1707199) 3.844627)))) (- (- (P-Position 25) (* (P-Position 25) 2.4853675)) (If (And (Or (P-Position 10) (P-Position 15)) (Or (< (P-Position 3) (P-Position 25)) (Not (Not (P-Position 1))))) (- 1.0983202 (- (+ (* (P-Position 25) 0.81480706) (P-Position 17)) (P-Position 7))) (- (* 1.3386335 (* (P-Position 25) (P-Position 25))) (- (P-Position 25) (* (P-Position 25) 2.4853675)))))))))))
```

74

Comparison of Backgammon Players

Player	% Wins vs. Pubeval
GP-Gammon-1	56.8
GP-Gammon-2	56.6
GP-Gammon-3	56.4
GP-Gammon-4	55.7
GP-Gammon-5	54.6
GP-Gammon-6	54.5
GP-Gammon-7	54.2
GP-Gammon-8	54.2
GP-Gammon-9	53.4
GP-Gammon-10	53.3

GP-Gammon-*i*: Our evolved players

75

Comparison of Backgammon Players

Player	% Wins vs. Pubeval
GP-Gammon-11	52.9
[Darwen, 2001]	52.7
GP-Gammon-12	51.4
GMARLB-Gammon [Qi & Sun, 2003]	51.2
GP-Gammon-13	51.2
GP-Gammon-14	49.9
GP-Gammon-15	49.9
GP-Gammon-16	49.0
GP-Gammon-17	48.1
GP-Gammon-18	47.8
ACT-R-Gammon [Sanner et al, 2003]	45.2
GP-Gammon-19	45.2
GP-Gammon-20	45.1
HC-Gammon [Pollack et al, 1997]	40.00

76

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…

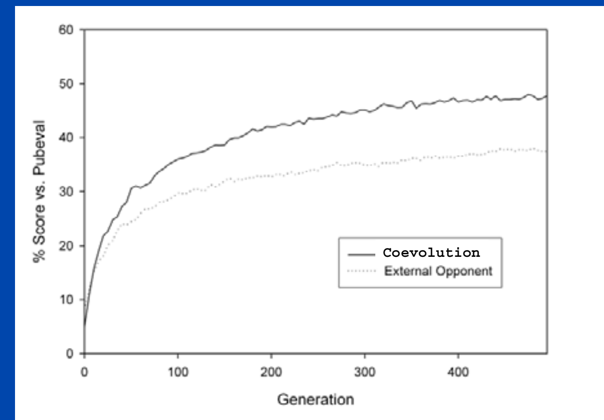
77

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

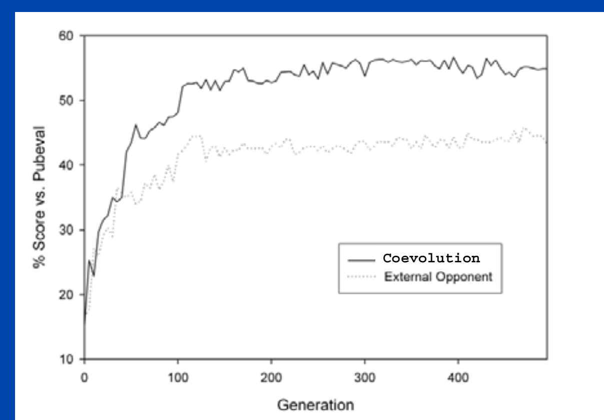
78

Compare Approaches: Average



79

Compare Approaches: Max



80

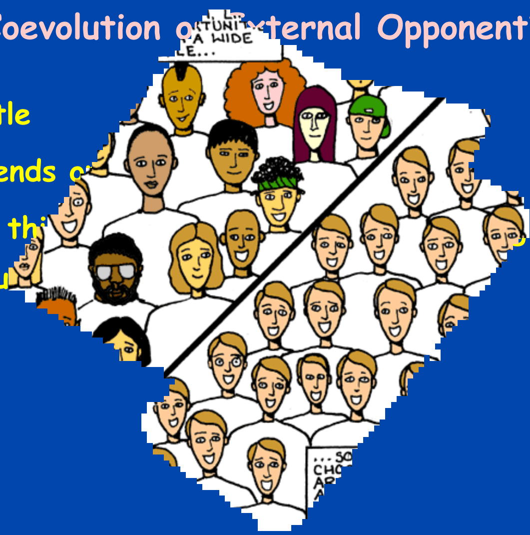
Coevolution is Better
(for backgammon)

- 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

81

So, Coevolution o xternal Opponent?

- Subtle
- Depends o
- One th
  evolu

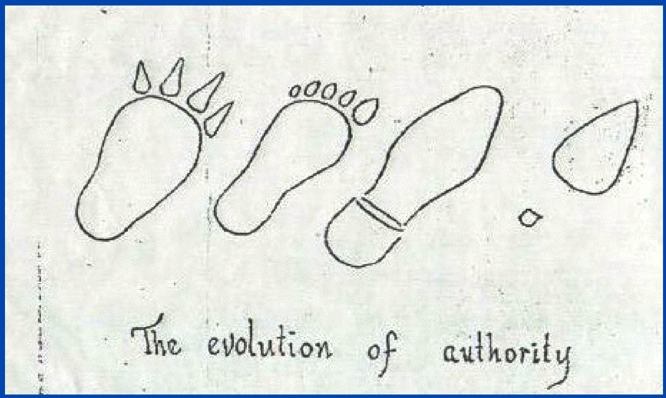


82

Observation

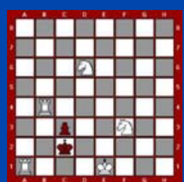
- Studying the GP-Gammon individuals, we concluded that strategy is mostly due to general query functions
- Specific query function helps "balance" strategy at critical moments

83



84

Chess



(Endgames)

85

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)

86

AI & Chess

- 1958: First chess AI program, novice level
- Over ~50 year… …dware gets …ally better
- Software…
- Well… Also
- 1997: Garry Kas… …champion, defeated by IBM's D…
- So…

87

AI & Chess

- 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

88

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

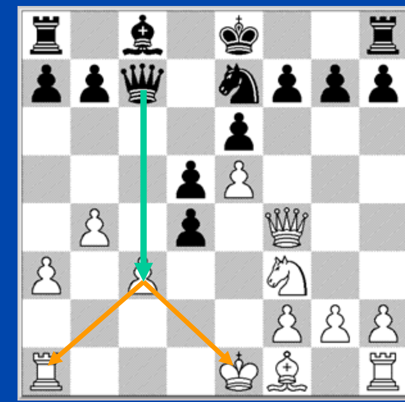
Chess Moves

- 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

values: 1 3 3/3.5 5 9 ∞

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

93

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…

94

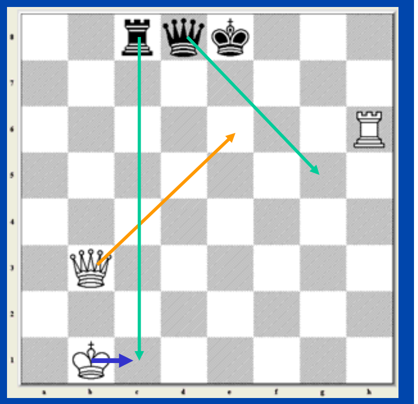
(Human) Grand Masters

- 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)

95

Endgame: Example

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



96

Endgames

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

97

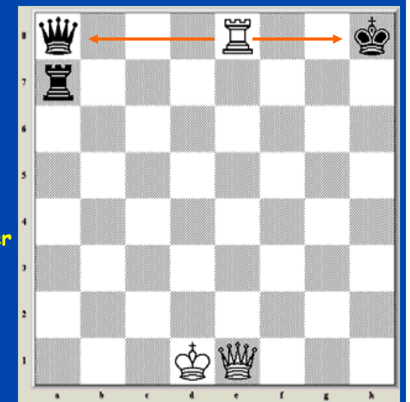
GP Building Blocks

- 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)

98

Example Feature: Fork

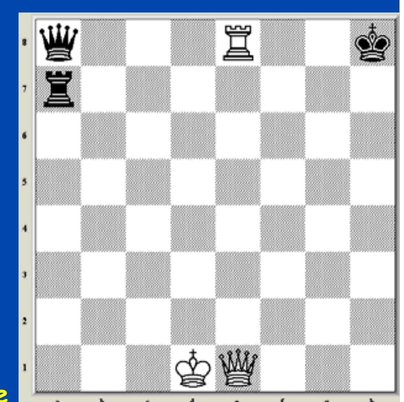
- My piece:
  - Attacking 2 or more pieces
  - Protected or not attacked
- Opponent's pieces:
  - unprotected
  - or, protected but of greater value
- Right: black must exchange Q for R



99

Fork: Traditional AI Search

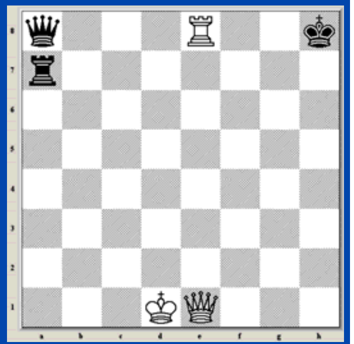
- Black: 3 legal moves
- Find that one of white's next moves (of 23) captures black queen
- Check all following moves for more piece exchanges
- Sometimes, still check other moves (non capturing)
- At end of search, compare remaining pieces



10

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



10

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

10

Genome Summary

Simple Terminals	Complex Terminals	Function Set
NotMyKingInCheck	EvaluateMaterial	If3(B,F,F)
IsOppKingInCheck	IsMaterialIncrease	Or2(B,B)
MyKingDistEdges	IsMate	Or3(B,B,B)
OppKingProximityToEdges	IsMateInOne	And2(B,B)
NumMyPiecesNotAttacked	OppPieceCanBeCaptured	And3(B,B,B)
NumOppPiecesAttacked	MyPieceCannotBeCaptured	Smaller(B,B)
ValueMyPiecesAttacking	IsOppKingStuck	Not(B)
ValueOppPiecesAttacking	IsMyKingNotStuck	
IsMyQueenNotAttacked	IsOppKingBehindPiece	
IsOppQueenAttacked	IsMyKingNotBehindPiece	
IsMyFork	IsOppPiecePinned	
IsOppNotFork	IsMyPieceNotPinned	
NumMovesMyKing		
NumNotMovesOppKing		
MyKingProxRook		
OppKingDistRook		
MyPiecesSameLine		
OppPiecesNotSameLine		
IsOppKingProtectingPiece		
IsMyKingProtectingPiece		

10

Fitness

- Success against peers
- Random-2-ways method: Each individual vs. fixed number of randomly selected peers (5)
- Scoring (based on chess tournaments):
  - Victory: 1
  - Material advantage (without mate): 0.75
  - Draw: 0.5
  - Loss: 0

10

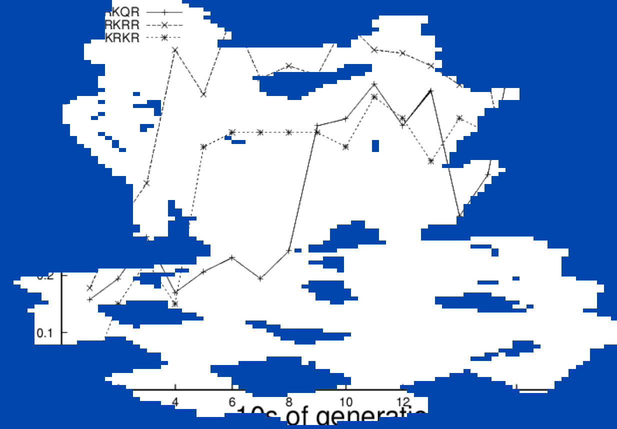
Two (Top) Opponents

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

10

Results: Master



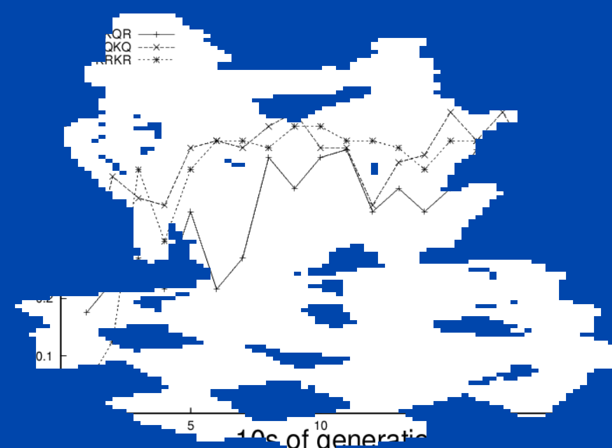
10

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

10

Results: CRAFTY



10

Sample GP-Endchess

Tree 0:
(If3 (Or2 (Not (Or2 (And2 OppPieceAttUnprotected NotMyKingInCheck) (Or2 NotMyPieceAttUnprotected 100*Increase))) (And2 (Or3 (And2 OppKingStuck NotMyPieceAttUnprotected) (And2 OppPieceAttUnprotected OppKingStuck) (And3 -1000*MateInOne OppKingInCheckPieceBehind NotMyKingStuck)) (Or2 (Not NotMyKingStuck) OppKingInCheck))) NumMyPiecesUNATT (If3 (< (If3 (Or2 NotMyPieceAttUnprotected NotMyKingInCheck) (If3 NotMyPieceAttUnprotected #NotMovesOppKing OppKingInCheckPieceBehind) (If3 OppKingStuck OppKingInCheckPieceBehind -1000*MateInOne)) (If3 (And2 100*Increase 1000*Mate?) (If3 (< NumMyPiecesUNATT (If3 NotMyPieceAttUnprotected -1000*MateInOne OppKingProxEdges)) (If3 (< MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne -1000*MateInOne NotMyPieceATT) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind)) NumOppPiecesATT) (If3 NotMyKingStuck -100.0 OppKingProxEdges))) (If3 OppKingInCheck (If3 (Or2 NotMyPieceAttUnprotected NotMyKingInCheck) (If3 (< MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne -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 -1000*MateInOne NotMyPieceATT) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind)) NumOppPiecesATT) -1000*MateInOne)) (If3 (< (If3 100*Increase MyKingDistEdges 100*Increase) (If3 OppKingStuck OppKingInCheckPieceBehind -1000*MateInOne)) -100.0 (If3 (And2 NotMyPieceAttUnprotected -1000*MateInOne) (If3 (< NumMyPiecesUNATT (If3 NotMyPieceAttUnprotected -1000*MateInOne OppKingProxEdges)) (If3 (< MyKingDistEdges #NotMovesOppKing) (If3 -1000*MateInOne -1000*MateInOne NotMyPieceATT) (If3 100*Increase #MovesMyKing OppKingInCheckPieceBehind)) NumOppPiecesATT) (If3 OppPieceAttUnprotected NumMyPiecesUNATT MyFork)))))
Tree 1:
(If3 NotMyPieceAttUnprotected #NotMovesOppKing 1000*Mate?)
Tree 2:
(If3 1000*Mate? NumMyPiecesUNATT -1000*MateInOne)

Multiple Endgames

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

	%Wins	%Advs	%Draws
Master	6	2	68
CRAFTY	2	4	72

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

Ami Hauptman and Moshe Sipper

Ben-Gurion University, Israel

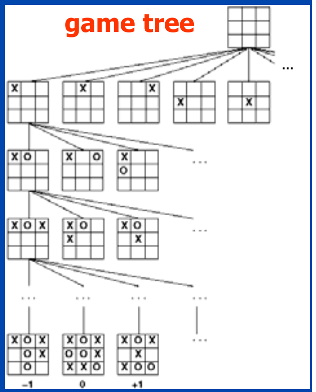
2007 "HUMIES" AWARDS FOR HUMAN-COMPETITIVE RESULTS

Monday, July 9, 2007

Genetic and Evolutionary Computation Conference GECCO 2007

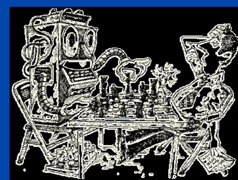
Game-Playing AI

- Game Strategy =
  Search + Knowledge
- Search:
  Number of nodes developed
- Knowledge:
  Evaluation of nodes
- Tradeoff between the two



Chess: Machine Players

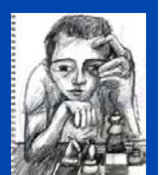
- Powerful contemporary engines
  - Crafty, Fritz, Deep Junior, …
  - Lots of search
  - Less knowledge
- Intelligent? Hmmm…
  - Very little generalization
  - Gobbles computational power
  - Deemed theoretically uninteresting [Chomsky, 93]



11

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



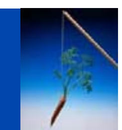
11

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

11

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

11

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

Mate-in	1	2	3	4	5
Depth in plies	2	4	6	8	10
Nodes developed	600	7K	50K	138K	1.6M

Major Result

Evolved search algorithm:

Number of nodes developed reduced by 47%

<u>with respect to world-class engine (not simple αβ)</u>

Mate-in	1	2	3	4	5
CRAFTY	600	7K	50K	138K	1.6M
Evolved	600	2k	28k	55K	850k

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

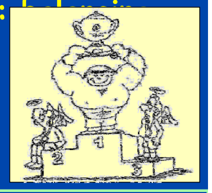
(G) result solves a problem of indisputable difficulty in its field

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

12

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

12

12

General Discussion

12

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

12

Cooperative with Humans

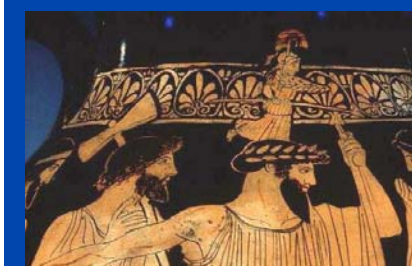
- GP readily accommodates human expertise
- Common AI view: Yada from Nada
- Like Athena, Greek goddess of wisdom, favorite child of Zeus, who had sprung fully grown out of her father's head
- Cooperative view: Man and machine work hand-in-keyboard

12

Attribute 17

Cooperative with Humans

12



VS.

12

Cooperative with Humans

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

GP + I → HC

Genetic Programming + (human) Intelligence

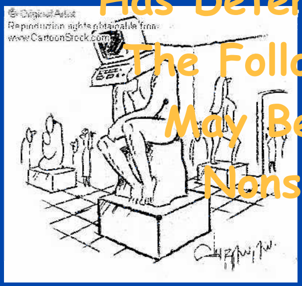
→ Human-Competitiveness

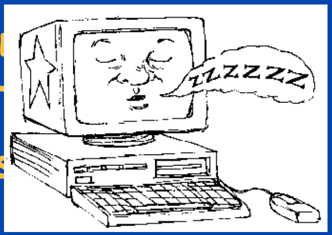
12

The Ultimate Goal…

Building a Strategizing Machine

Warning:
The Surgeon General
Has Determined That
The Followi
May Be T
Nonsens





13

A/I or A-I ?

- Importance of high "artificial-to-intelligence ratio" (Koza *et al.* , 2003): Maximize A/I.
- Better: A - I ≥ M_{ε} (meaningful epsilon)
- Pore in as much I as possible, with goal of maximizing (machine's) added Value: A

13

A Final Thought…

Machines Might Be

Human-Competitive

13

BUT…

13

NO!

13

Are Humans

Machine-Competitive?

13

Human Cellular Automata

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

13

Human Cellular Automata

- Slow
- Error-prone
- Processors tend to complain

13

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*

13

Speaking of (brilliant) students…

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

13

14

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
  IEEE SMC-C, vol. 37, no. 4, pp. 583-593, July 2007
- A. Hauptman and M. Sipper
  EVOLUTION OF AN EFFICIENT SEARCH ALGORITHM FOR THE MATE-IN-N PROBLEM IN CHESS
  Proceedings EuroGP2007, pp. 78-89

14