PCG: Search Revisited, Evolution, and More

2016-07-14

Questions

1. How can we describe decision making?
2. What do the algorithms we've seen share?
3. What are the dimensions we tend to assess?
4. FSMs/Btrees: ____ :: Planning : _____
5. For the 2nd blank, we need m_____s.
6. When is reactive appropriate? Deliberative?
7. What is the 'hot-potato' passed around (KE)?
8. H_____ have helped in most approaches.
9. Which approach should you use?

Questions

1. What are the 2 most “complex” decision making techniques we’ve seen?
2. What are their strengths? Weaknesses?
3. What is the key (insight) to their success?
4. What is typically necessary to support this insight (hint: used in Planning + RBS)?
5. What does Planning have that (forward chaining) RBS do not?
6. When do we need a communication mechanism?

PCG Questions

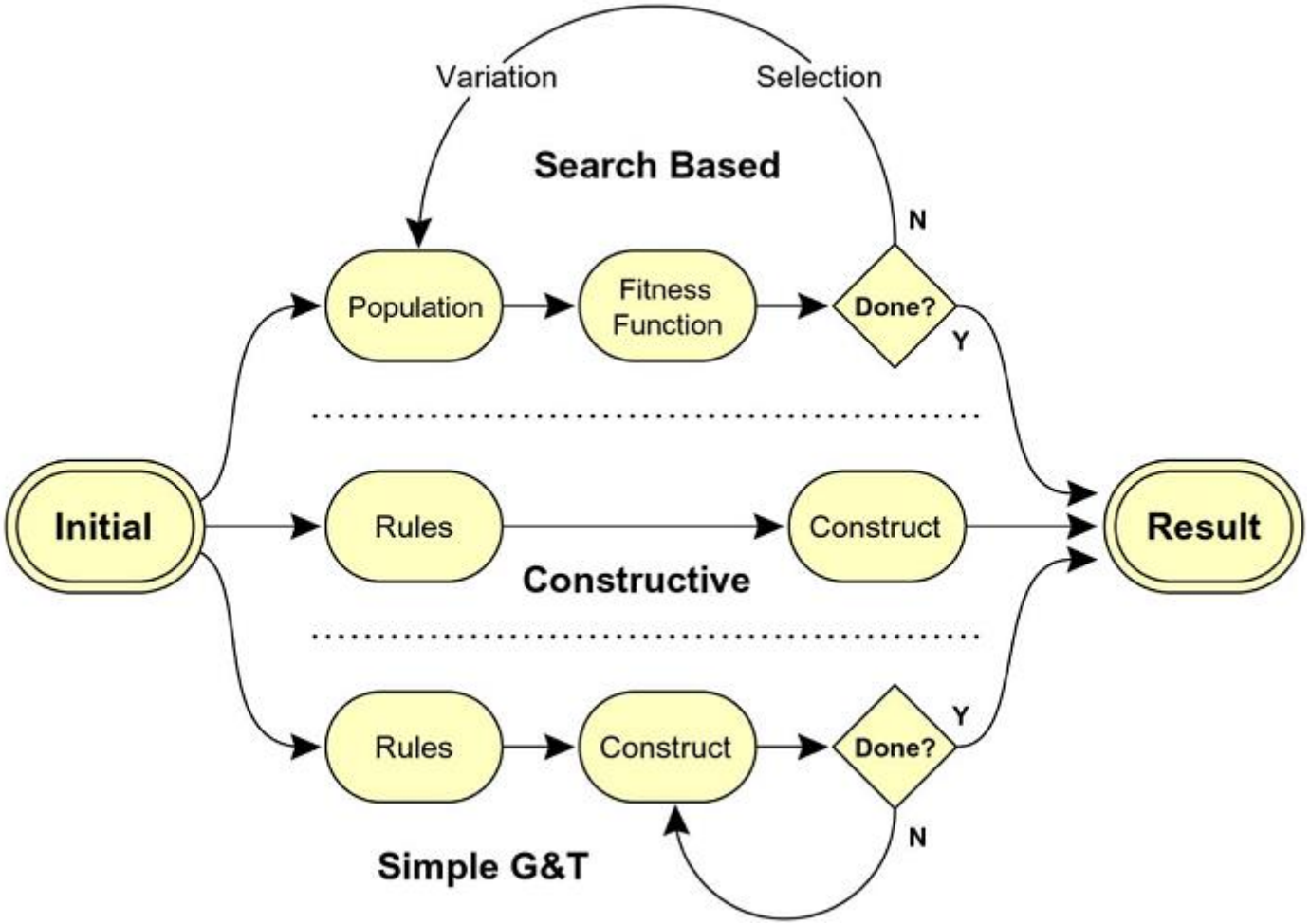
1. What is PCG?
2. PCG can be used to pr____ or ad____ game aspects
3. Why does industry care about PCG?
4. What are some risks of PCG?
5. Major concerns involving PCG include…
6. What is a player model? What does it allow?
7. What are ways to get a player model?
8. Bartle's 4-part feature vector: <k,a,e,s>

PCG Concerns

- Speed (real-time/design time)
- Reliability (catastrophic failures/crashes)
- Controllability (wrt constraints and goals)
- Diversity (variations on a theme)
- Creativity (looks “computer-generated”)

See IGDA Webinar, 10 December 2014: PCG in games

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172-186.



http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5756645&tag=1

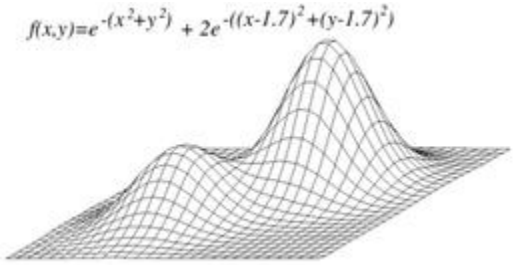
PCG AS PARAMETER SEARCH: HILL CLIMBING

Start Simple

- Magic numbers are everywhere
- Parameter modification
  - Find/calculate value of 1+ parameters
  - Aim: find best values of parameter
- Graph: Fitness/Quality vs parameters
  - “landscape”
  - Energy vs fitness
- Assume f(params) → fitness

Hill Climbing

- https://en.wikipedia.org/wiki/Hill_climbing
- Guess, check, modify parameter value, check…
- What direction to change parameter?
  - Change one param at a time, check score
  - Move up steepest gradient
  - Assume fixed step size
- "Parameter optimization"
- Simple, fast, can give good results
- Problems?



Extensions to Hill Climbing

- Local (sub)optimality, search gets “stuck”
  - The more local maxima, more difficult to solve
  - At worst, fitness is random/not correlated to nearby values
  - Step back. Is goal to find “optimal”? “Satisficing”
- Fixes
  - Momentum (record prev score improvements)
  - Adaptive resolution (think granularity)
  - Multiple trials (initial guesses)
  - Annealing (add term to rep temperature)

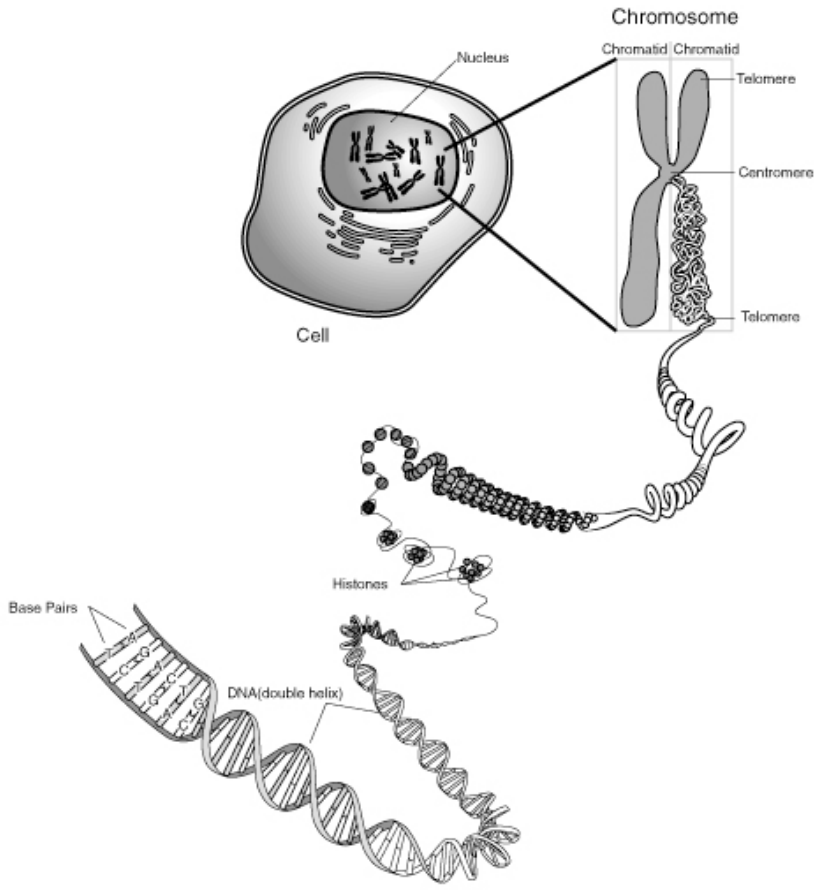
PCG AS PARAMETER SEARCH: GENETIC ALGORITHMS

Video Examples

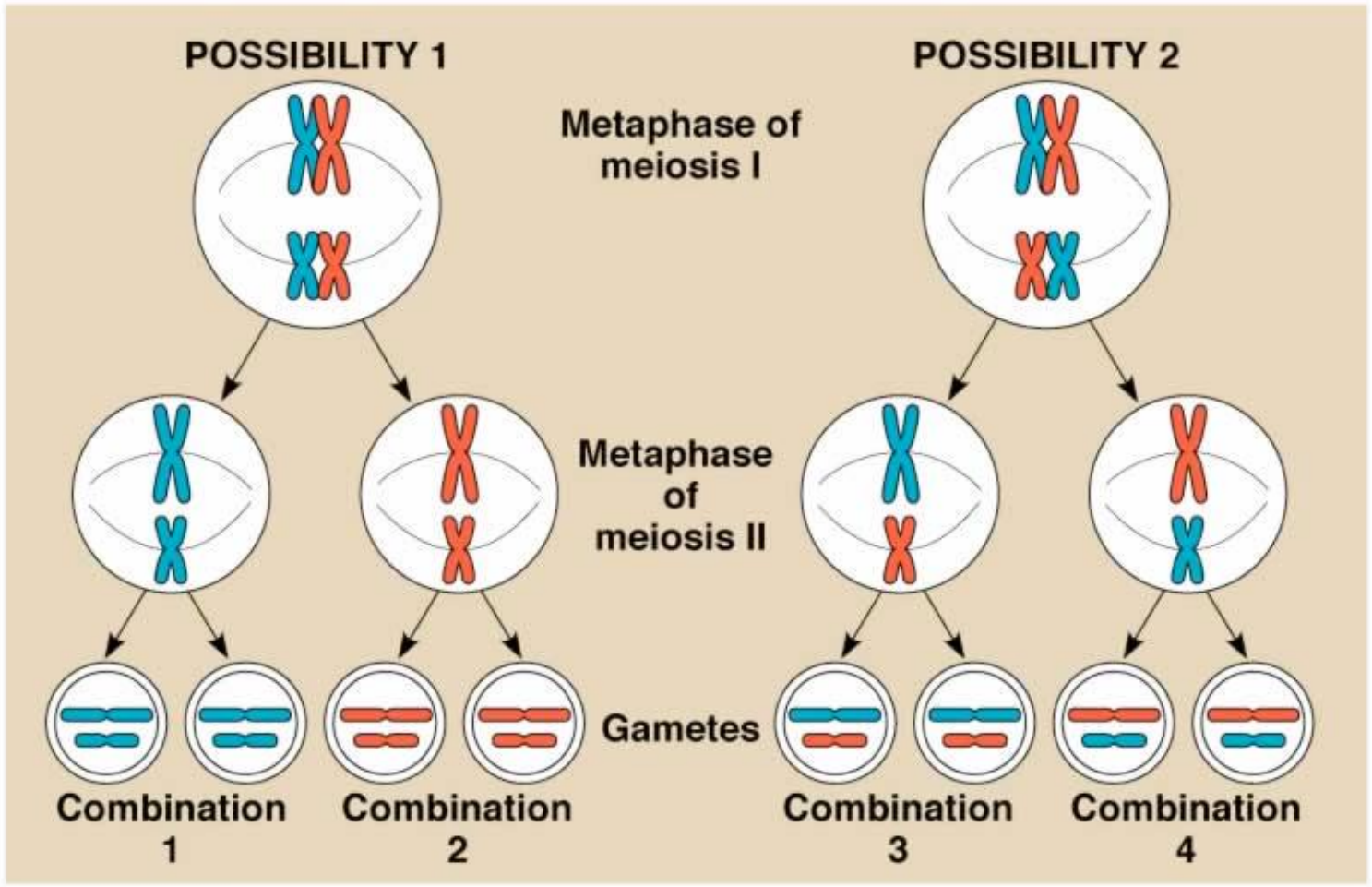
- Evolving Faces
- Evolving Virtual Creatures (Karl Sims)
- Evolving muscle-based bipedal locomotion
- Evolving Artificial Creatures
- Killer Fish

Biology Metaphor

"Variation is a feature of natural populations and every population produces more progeny than its environment can manage. The consequences of this overproduction is that those individuals with the best genetic fitness for the environment will produce offspring that can more successfully compete in that environment. Thus the subsequent generation will have a higher representation of these offspring and the population will have evolved.“
-Darwin



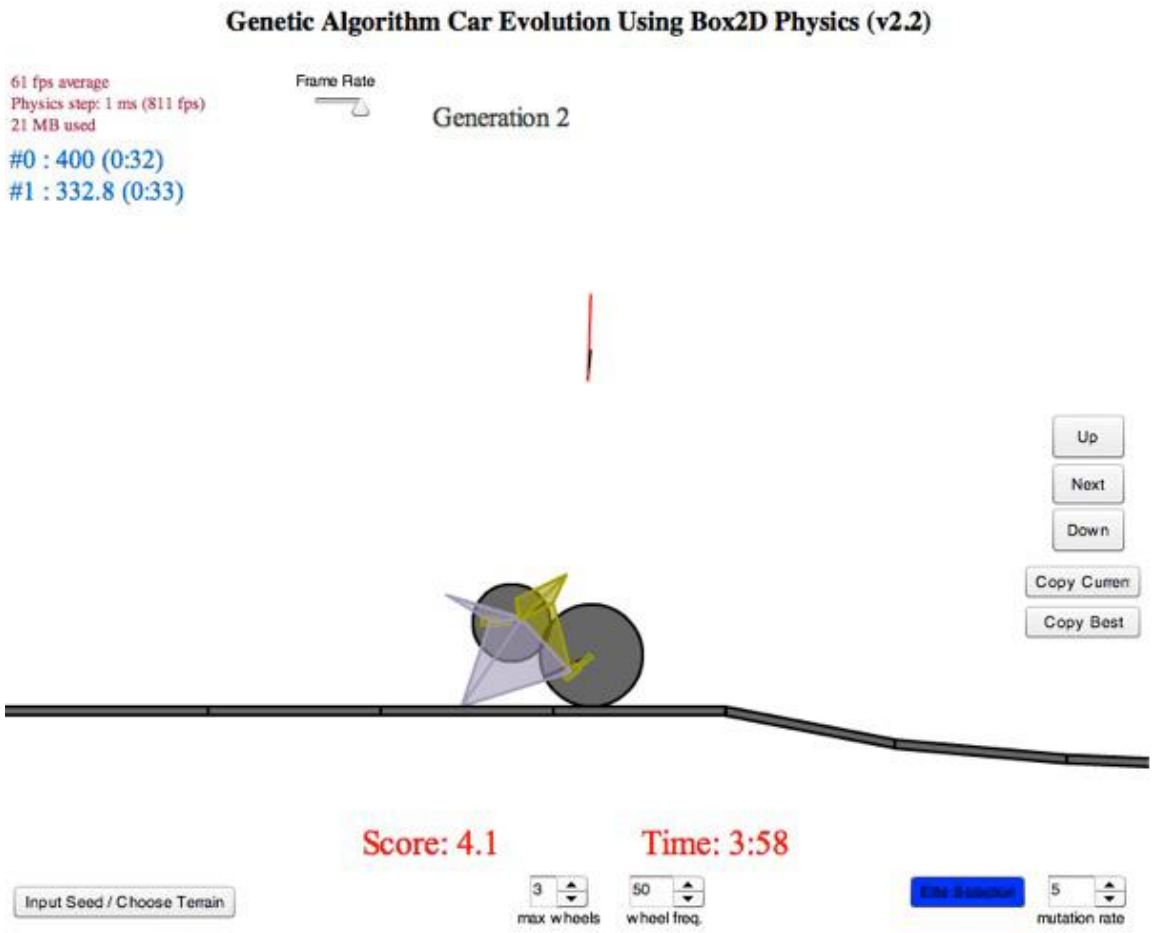
Recombination



The GA Problem

- 1st Step: Encoding a solution to your problem (i.e. a *chromosome*)
  - 10101010000101011101010101101O
  - e.g. values for coefficients in a function, numeric preferences for selecting rules, or weights in your FSM or BT
- Not human readable

Evolving a Car



[BoxCar2d]

Steps

1. Create a random set of *n* chromosomes [“population”]

Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem

Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem
3. Assign a fitness score to each tested chromosome

Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem
3. Assign a fitness score to each tested chromosome
4. Remove the *m%* (*m* < 100) worst chromosomes

Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem
3. Assign a fitness score to each tested chromosome
4. Remove the *m%* (*m* < 100) worst chromosomes
5. Cycle through selected pairs of chromosomes and *cross-over*

Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem
3. Assign a fitness score to each tested chromosome
4. Remove the *m%* (*m* < 100) worst chromosomes
5. Cycle through selected pairs of chromosomes and *cross-over*
6. Randomly mutate during cross-over

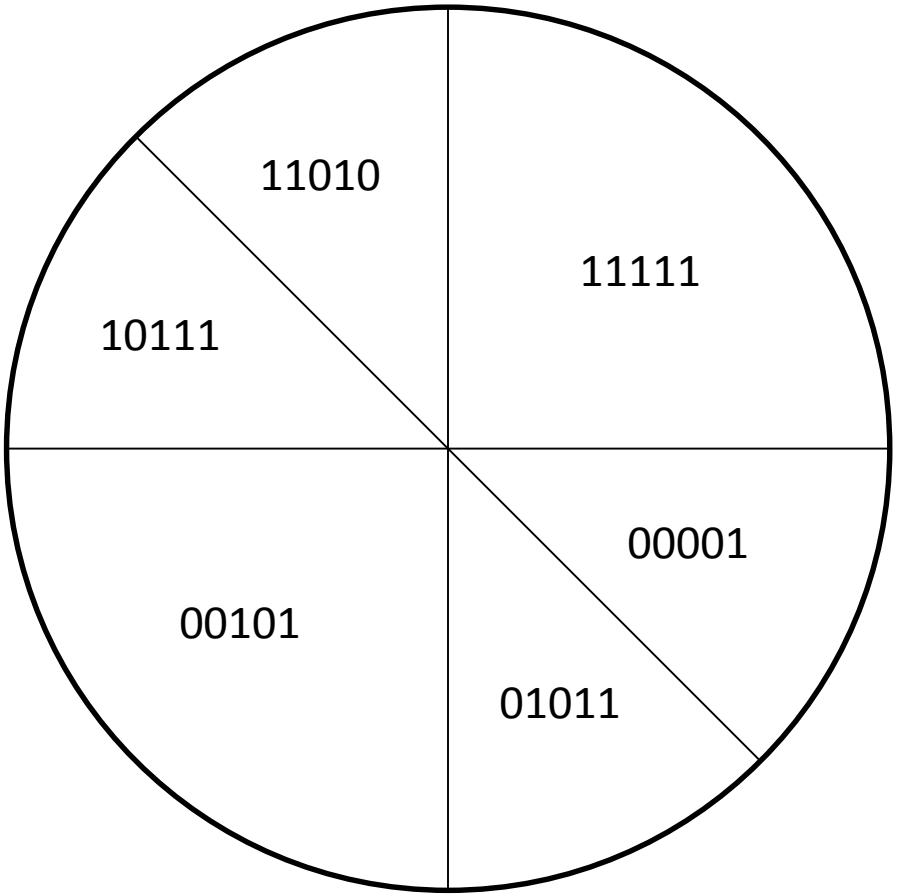
Steps

1. Create a random set of *n* chromosomes
2. Test each chromosome to see how good it is at solving the problem
3. Assign a fitness score to each tested chromosome
4. Remove the *m%* (*m* < 100) worst chromosomes
5. Cycle through selected pairs of chromosomes and *cross-over*
6. Randomly mutate during cross-over
7. Repeat steps 2-6 until optimality is reached (local or global?)

Population

- How many individuals (or chromosome representations) you have in the gene pool
- How many should you have? How many should you reproduce?
  - Small pop.: Risk replacing individuals before reproduction; Less diversity
  - Large pop.: More diversity; converge fast initially, but little progress later

Roulette Wheel Selection



Scaling

- Used in combination with Roulette Selection
- Helps prevent premature convergence

- Rank scaling
  - Starting with raw fitness scores, convert to rank listing
  - Use ranks for roulette proportions

  e.g. [235, 123, 54, 45, 32] becomes [5,4,3,2,1]
  - Especially in early generations, to help prevent premature convergence
  - Good because fitness scores early on may be too widespread

Scaling

- Used in combination with Roulette Selection
- Helps prevent premature convergence

- Rank scaling
- Sigma scaling (σ is std dev of fitnesses)

  NewFit[i] = (RawFit[i] – AvgFit) / 2σ

  – Keeps selection pressure (diversity) constant over many generations.

Scaling

- Used in combination with Roulette Selection
- Helps prevent premature convergence

- Rank scaling
- Sigma scaling
- Boltzmann scaling

  NewFit[i] = (RawFit[i]/Temperature) / Avg(RawFit/Temperature)

  Temperature is found through trial and error

  e.g. T = 3*pop_size – 0.05*numGenerations

  – Keeps selection pressure low at beginning and high at the end.
  – As alg converges, fitter individuals are given a preference (High selection pressure = low diversity)

Tournament Selection

- Select n% of the population (typically 2<n<10)
  - As n decreases, diversity increases
- The most fit member of the group is used for crossover
  - Alternately, crossover the 2 most fit members of the group
- Faster than roulette selection

Elitist Selection

- Select the *n%* best individuals from the population and advance them *unchanged* to the next generation
- Typically, $1 < n < 10$; potentially $n < 20$
- Too much elitism ---> early convergence

Kiss: μ + λ

- Create a population of μ + λ individuals
- Each generation
  - Evaluate all individuals in population
  - Sort by fitness
  - Remove the worst λ individuals
  - Replace those removed with mutated copies of the μ best (no crossover)

Crossover

- Given crossover rate and format, swap bits of the parents

  e.g. 100111 and 101001 would potentially yield:

Crossover

- Given crossover rate and format, swap bits of the parents

  e.g. 100111 and 101001 would potentially yield:

  100001 and 101111 (One-point crossover)

Crossover

- Given crossover rate and format, swap bits of the parents

  e.g. 100111 and 101001 would potentially yield:

  100001 and 101111 (One-point crossover); or…

  101011 and 100101 (Two-point crossover)

Crossover

- Given crossover rate and format, swap bits of the parents

  e.g. 100111 and 101001 would potentially yield:

  100001 and 101111 (One-point crossover); or…

  101011 and 100101 (Two-point crossover); or…

  100111 and 101001 (Random point selection)

Crossover

- Given crossover rate and format, swap bits of the parents

  e.g. 100111 and 101001 would potentially yield:

  100001 and 101111 (One-point crossover); or…

  101011 and 100101 (Two-point crossover); or…

  100111 and 101001 (Random point selection)

- Multi-point crossover
  - Select each gene from either parent, possibly ensuring that an equal number come from each parent

Mutation Rate

- Small chance of a bit being flipped
- 101111 becomes 101110

Niching

- Method for retaining diversity
- Useful when environment might have multiple peaks
- Also good for protecting a new innovation within a population

- Explicit Fitness Sharing

  NewFit[i] = OldFit[i] / NumNeighbors

Niching

- Method for retaining diversity
- Useful when environment might have multiple peaks

- Explicit Fitness Sharing
- Speciation
  - Requires crossover only occurs within “breeds”
  - Species are killed when pop = 0 or fitness hasn’t increased over several generations
  - Might want to try higher mutation rates

Simple Example

Problem:

Given the digits 0 through 9 and the operators +, -, *, and /, **find a sequence that will represent a given target number**. The operators will be applied sequentially from left to right as you read and any extraneous information will be ignored.

Encoding

0:	0000	7:	0111
1:	0001	8:	1000
2:	0010	9:	1001
3:	0011	+:	1010
4:	0100	-:	1011
5:	0101	*:	1100
6:	0110	/:	1101

Example Solution

23 = 6 + 5 * 4 /2 + 1

--->

0110 1010 0101 1100 0100 1101 0010 1010 0001

6 + 5 * 4 / 2 + 1

Deciding on a Fitness Function

- Hardest part of the process
- e.g., “inverse proportional to the difference between the solution and the chromosome’s value”
- e.g., with a target of 42 and a value of 23
  - Fitness: 1/(42-23) = 1/19
  - 1/0 ---> success

Using Real Values

- In games, we may want our genomes to use real floating-point numbers instead of bit strings
- Allows us to exploit mathematical properties of a landscape

Michalewicz Method

- Options for mutation and crossover that take advantage of real-value encoded GAs

Michalewicz Mutation

- Boundary Mutation
  - With probability *p*, change a gene to G_{min} or G_{max}

Michalewicz Mutation

- Boundary Mutation
- Replace Mutation
  - With probability *p*, set gene to a uniform random number in $[G_{min}, G_{max}]$

Michalewicz Mutation

- Boundary Mutation
- Replace Mutation
- Non-Uniform Mutation
  - With probability *p*, adjust a gene by some small random amount *k*
  - Decrease range of *k* over time

Michalewicz Crossover

- Arithmetical Crossover
  - G_{child} = average(G_1, G_2)

Michalewicz Crossover

- Arithmetical Crossover
- Simple Crossover

Michalewicz Crossover

- Arithmetical Crossover
- Simple Crossover
- Heuristic Crossover
  - $G_{child} = r(G_2 - G_1) + G_2$
  - r is random number in [0, 1]
  - G_2 is fitter parent

Tuning Parameters

- Population size
- Number of generations
- Fitness function
- Representation
- Mutation rate
- Crossover operations
- Selection procedure
- Number of solutions to keep

Large pop takes too long. Small pop doesn't search a large enough space, and converges to poor soln.

Too much mutation leads to random search. Not enough, then we lose diversity and stagnate.

Criticisms

- Repeated fitness evaluation may be expensive
- Unclear stop criterion
- Do not scale with complexity
- Prone to local, rather than global, maxima

GA Applications

- Applications anywhere with a large search domain
  - Especially where traditional search or optimization would be slow
- Useful when:
  - Domain knowledge is scarce
  - Hard to encode expert knowledge
  - No mathematical analysis available
- Best used offline

GAs and FPS Games

- Counterstrike methodology
  - Select parameters to tune
  - Allow the GA to evolve the parameters
  - Pit the evolved bots against hand-tuned bots
- Fitness function: $ earned
- GA bots performed as well as hand-tuned bots

GAs and RTS Games

- Tune AI strategy to target human player weaknesses
  - Tune parameters that define AI personality (e.g. unit preference, scientific advance preference, offense vs. defense, etc.)
- Tune behavior of individuals or groups of units

GAs and Game AI

- GAs seek optimal solutions. Is that what we want from Game AI?

GAs and Game AI Research

- Counterstrike
- FreeCiv
- Quest Generation

GAs in Review

- Have problems with local maxima
  - Niche penalty can avoid this sometimes
- GAs can rapidly find *good* solutions in general
- Problem domains
  - Scheduling and timetabling
  - Engineering // optimization problems
- Not widespread in gamedev, but has been used in conjunction with other ML techniques

PCG See also

- Papers linked above & T-square
- IGDA Webinar, 10 December 2014: PCG in games: perspectives from the ivory tower
  - https://www.youtube.com/watch?v=UVRqCK6m7m4
- PCG Book http://pcgbook.com/
  - Grammars: Chapter 5 http://pcgbook.com/wp-content/uploads/chapter05.pdf
- 9.1: Genetic Algorithms and Evolutionary Computing - The Nature of Code
  - https://www.youtube.com/watch?v=6l6b78Y4V7Y