PCG: Search Revisited, Evolution, and More

2018-03-29

Satisficing is a decision-making strategy or cognitive heuristic that entails searching through the available alternatives until an acceptability threshold is met.[1] The term *satisficing*, a combination of *satisfy* and *suffice*,[2] was introduced by Herbert A. Simon in 1956,[3] although the concept was first posited in his 1947 book *Administrative Behavior*.[4][5] Simon used satisficing to explain the behavior of decision makers under circumstances in which an optimal solution cannot be determined. He maintained that many natural problems are characterized by computational intractability or a lack of information, both of which preclude the use of mathematical optimization procedures. He observed in his Nobel Prize in Economics speech that "decision makers can satisfice either by finding optimum solutions for a simplified world, or by finding satisfactory solutions for a more realistic world. Neither approach, in general, dominates the other, and both have continued to co-exist in the world of management science".

https://en.wikipedia.org/wiki/Satisficing

PCG high-level Methods

- Search
  - Hill Climbing
  - Simulated Annealing
  - Genetic Algorithms
- Rule systems
- Generative Grammars
- Constraint Solving

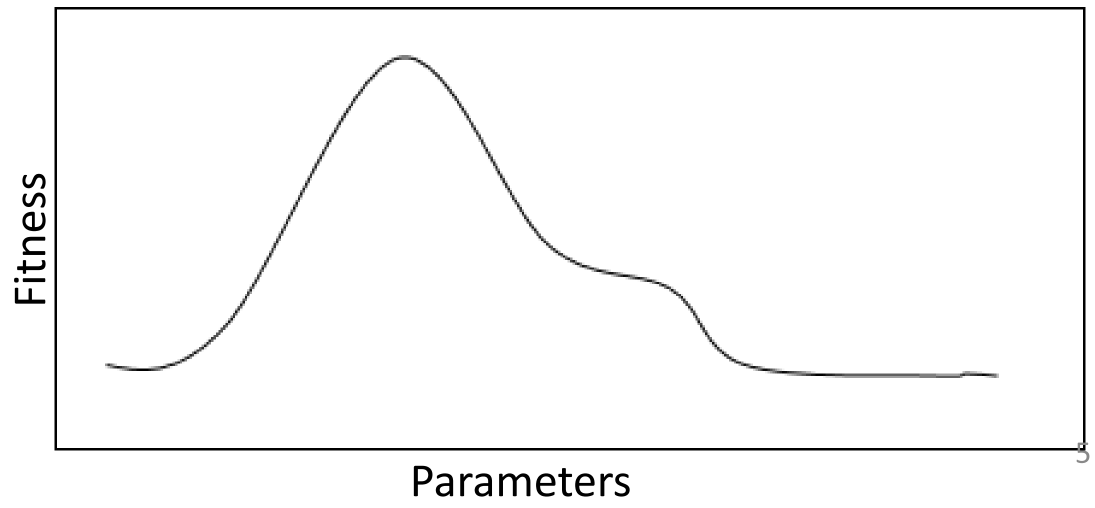
PCG AS PARAMETER SEARCH: HILL CLIMBING

Hill Climbing (Review?)

- From some generated element we can get to “neighbors”, which allows us to define a space
- The quality of any element can be derived by a heuristic to get some value
- **Local Maxima**: A point better than all its neighbors
- **Global Maxima**: A point better than all other points
- **Hill Climbing**: From random start point, pick the best neighbor (equivalent to greedy search)

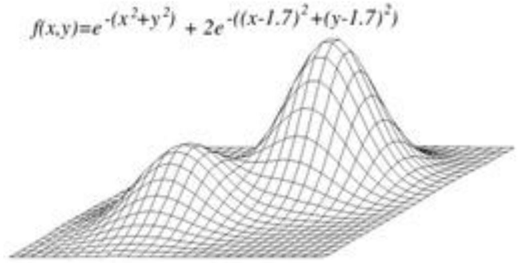
Start Simple

- 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



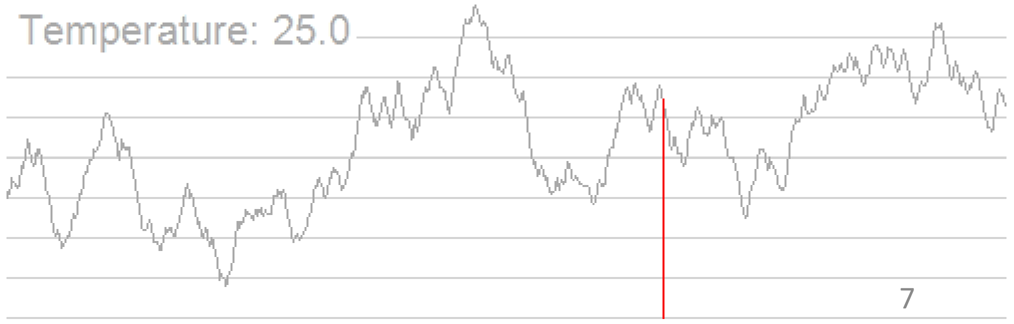
Hill Climbing

- 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)



Simulated Annealing (Review)

- Find approximate to the/a global optimum
- Inspired by annealing in metallurgy

- Pick a neighbor based on the some probability *T* (based on how long the algorithm has been running) and the quality of that neighbor

Simulated Annealing (Review?)

```
state= randomly choose a state

for k = 0 through maxTimeSteps:
    T = calculateTemperature(k/maxTimeSteps)
    newState = randomSelect(neighbors(state))
    if P(E(state), E(newState), T)>random(0,1)):
        state = newState
return state
```

Simulated Annealing

- Can give good results!
- Highly dependent on the temperature function, which is domain (and even problem) dependent
- At worst it is basically random search, with all those associated fallbacks

PCG AS PARAMETER SEARCH:
GENETIC & EVOLUTIONARY ALGORITHMS

Video Examples

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

Genetic Algorithms

- Loosely inspired by Darwin’s Theory of Evolution
- Sometimes called “Evolutionary search”
- Consistently pretty popular (if only for the tagline “evolving artificial intelligence to solve…”)

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
- See also list of applications: https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications

Genetic Algorithm Pseudocode

```
population = set of random points of size *X*
time = 0
while E(population)<*threshold* and time<max:
        time ++
        **Mutate**(population)
        population = **Crossover**(population)
        population = **Reduce**(population)
return bestE(population)
```

GA Pseudocode cont

- **Mutate**: Given some probability, randomly replace a member of the population with a neighbor
  - Make a random change
- **Crossover/recombination**: Take pairs of the initial population (chosen based on fitness), and combine their features randomly till population grows up to size Y (where $X<Y$)
- **Reduce**: Reduce the size of the population back down to X

Genetic Algorithms

Pros

- Middling authorial burden (more than hill climbing, less than generative grammars/rule system)
- High likelihood of finding global optima-ish

Cons

- Takes skill to pick and balance mutation/crossover
- Over-reliance

Search-based PCG

Pros

- Allows for designers to specify high-level desires in heuristics rather than low level content authoring

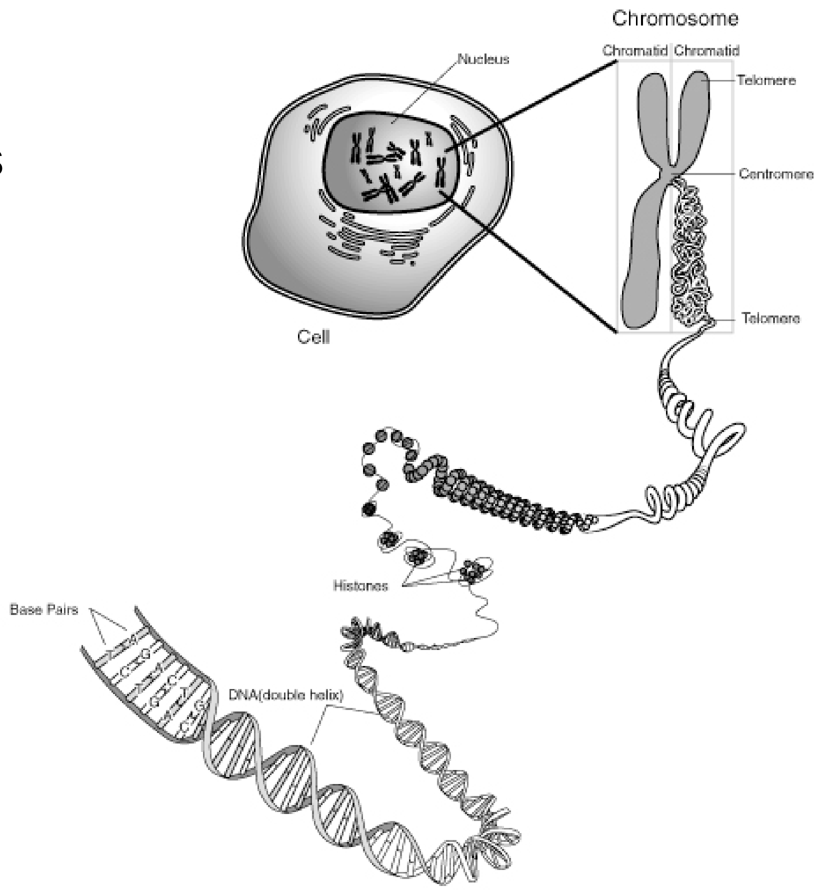
Cons

- Coming up with a good heuristic is hard
- Many devs will prefer to just use generative grammars

INFINITE VARIETY

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



The GA Problem

- 1st Step: Encoding a solution to your problem (i.e. a *chromosome*)
  - 101010100001010111010101011010
  - e.g. values for coefficients in a function, numeric preferences for selecting rules, or weights in your FSM or BT
- May not be human readable

Steps

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

   Each chromosome represents an individual or a state or a particular agent

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. Reduce: 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. Reduce: Remove the m% (m < 100) worst chromosomes
5. Cross-over: 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. Reduce: Remove the m% (m < 100) worst chromosomes
5. Cross-over: Cycle through selected pairs of chromosomes and cross-over
6. Mutate: 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. Reduce: Remove the m% (m < 100) worst chromosomes
5. Cross-over: Cycle through selected pairs of chromosomes and cross-over
6. Mutate: Randomly mutate during cross-over
7. Repeat steps 2-6 until optimality is reached (local or global?)

REDUCE / SELECTION

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

Kiss: $\mu + \lambda$

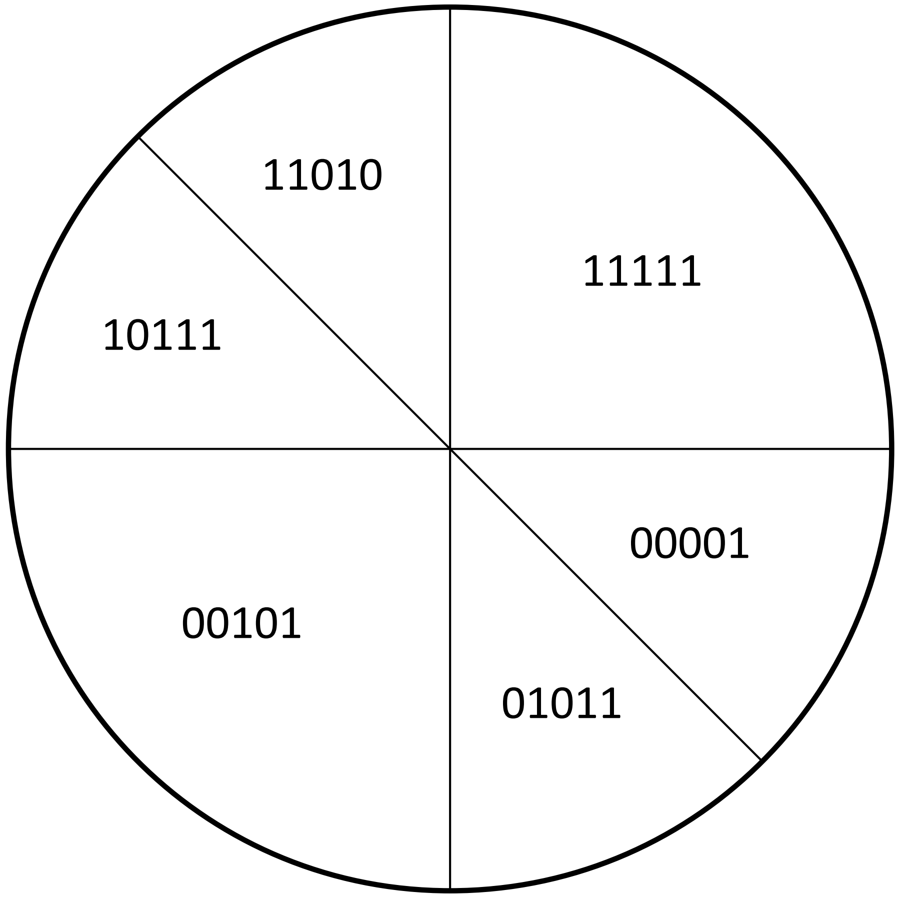
- Create a population of $\mu + \lambda$ individuals
- Each generation
  - Evaluate all individuals in population
  - Sort by fitness
  - Remove the worst λ individuals
  - Replace those removed with mutated copies of the μ best (no crossover)

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

Generally, you want some elitism regardless of your other selection techniques.

Roulette Wheel Selection



This is a way of choosing members from the population of chromosomes **in a way that is proportional to their fitness**

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

CROSSOVER

Crossover

Having a good crossover function that fits the structure of your problem representation is crucial.

- 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 & FITNESS

Mutation Rate

- Why do we have crossover and mutation?
  - Crossover explores faster. Mutation exploits local region.
  - You ALWAYS need to have mutation
- What is the benefit of mutation?
  - Making random changes to our solutions helps get us out of local maxima

- 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

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

GAs and Decision Making

- GAs can be used to tune parameters in rules and FSMs

```
if (enemy.distance <= 5)
        ATTACK-WITH-KNIFE()
else if (enemy.distance > 5 AND enemy.distance <=30)
        ATTACK-WITH-SUBMACHINE-GUN
else ATTACK-WITH-RIFLE
```

GAs and Decision Making

- Counterstrike example
  - 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

Using a Genetic Algorithm to Tune First-Person Shooter Bots. Nicholas Cole, Sushil J. Louis, and Chris Miles. 2004. https://pdfs.semanticscholar.org/24ec/f958b25b791b63271774b3e115ad38185015.pdf

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

Ponsen, Marc, and Pieter Spronck. *Improving adaptive game AI with evolutionary learning*. Diss. Masters Thesis, Delft University of Technology, 2004.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.6055&rep=rep1&type=pdf

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

- 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