Genetic Algorithm for Optimization

Prof. Alan D. Freed

J. MIKE WALKER '66 DEPARTMENT OF
MECHANICAL ENGINEERING
TEXAS A&M UNIVERSITY

afreed@tamu.edu

MEEN 357, Spring 2021

2021

Outline

- Why Use Genetic Algorithms?
- Case Study: Apollo 13 Power-Up of Odyssey for Reentry
- Features of Genetic Algorithms
- Class Interfaces for:
  - Genes
  - Chromosomes
  - Genome
  - Creatures
  - GenAlg: the Driver
  - Species: the User Interface
- An Example: Parameterization of a Creep Model

Goldberg, D. E., *Genetic Algorithms in Search, Optimization & Machine Learning*, 1989, Addison Wesley.

Genetic Algorithms

The Most Important Goal of Optimization is Improvement

A. D. Freed

Optimization is a search for 'improvement'. 'Better' is often preferred to 'best'. Improvement can only be determined once we know how to measure and adjust between 'good' and 'bad'.

Calculus based searches are used when one's objective function is well behaved and one has a good 'guess' for a solution. Their searches have local span over a plausible solution domain.

Uninformed random searches are good for exploring the whole solution domain, but they are subject to the curse of dimensionality whenever they are implemented and applied.

Question: How can one 'inform' an otherwise random process by adding 'intellegence' to its searching algorithm? Look to nature.

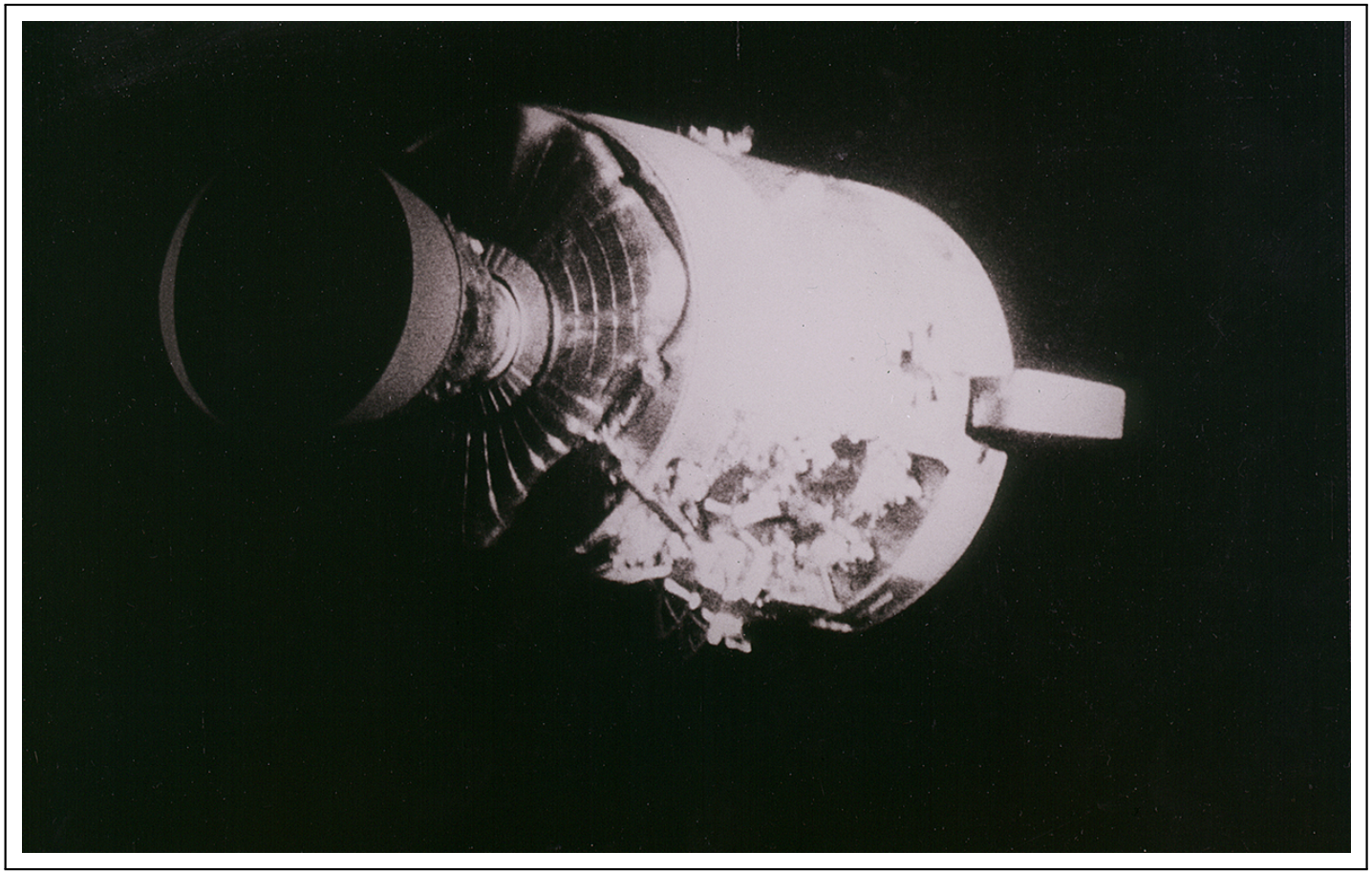
Genetic algorithms use random choice to guide a search into regions of the search space with likely improvement using probabilistic events that mimic processes of natural selection.

There are many other 'nature inspired' search techniques.

Case Study: Apollo 13, April 17, 1970

Power-Up Sequence of the Odyssey Command Module

The damaged Service Module of Apollo 13, a.k.a. Aquarius.



Lovell to Houston: "Farewell, Aquarius, and we thank you."

Case Study: Apollo 13, April 16, 1970

Power-Up Sequence of the Odyssey Lunar Module

Command Module "Odyssey" had been powered down for just over 3 days to conserve power. During that time the crew (Jim Lovell, John Swigert and Fred Haise) moved into the Lunar Module (or Lem) using it as a lifeboat. (What problem arose?)

To return to Earth, the Crew had to power-up Odyssey for navigation and control. The issue was how to sequence the powering up of devices so as not to deplete available power in the batteries prior to splash down. NASA worked this problem over the 3 days that Apollo 13 returned to Earth. A solution was obtained by trail and error fueled through desperation.

Could one use a genetic algorithm to determine an optimal outcome? It needn't be the best solution, it just has to be a viable solution. (NASA went to the Moon using slide rules.)

Switch Positions in Power Up

A Genetic Interpretation

A. D. Freed

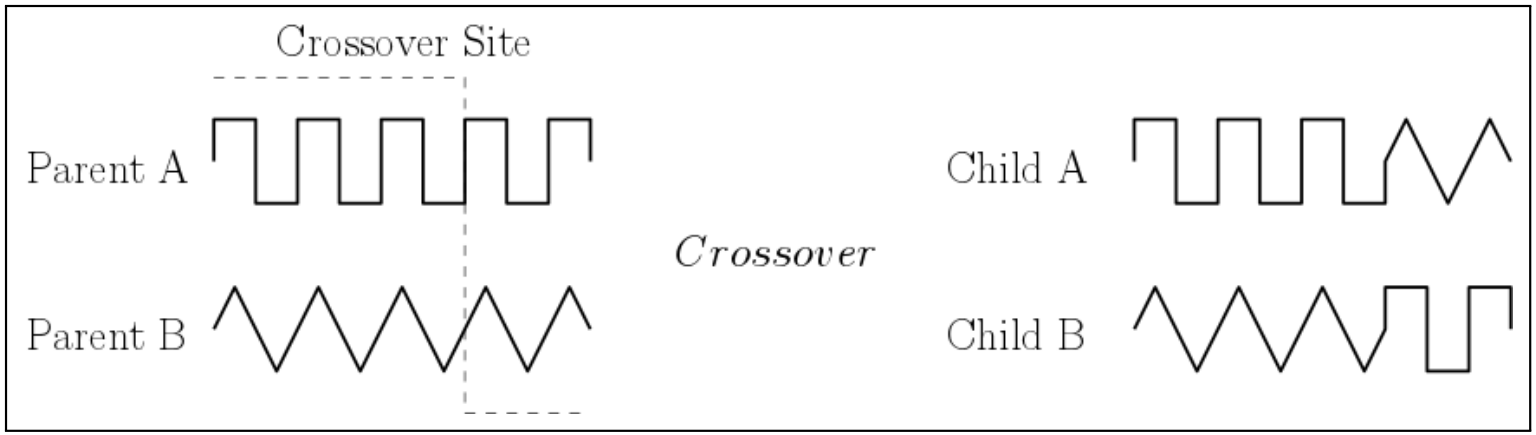
Consider five switches, say. Represent those that are switched on as '1' and those that are off as '0'. These switches and their states can then be represented as a simple string, e.g., 10011. These settings associate with a certain power consumption, i.e., a cost function. We seek a set of settings whose power consumption is less than the available power. Genetic algorithms are ideal for such problems.

Genetic Algorithms Have the Following Features:

1. Encode parameters into strings (chromosome = string of genes), e.g., 10011, and work with these strings, not the parameters themselves. A dominant gene is 1; a recessive gene is 0.
2. Explore from a population of points (creatures—each creature has an unique genome; a genome = string of chromosomes), rather than searching from a single point.
3. To advance, search by random sampling and select by fitness (via an objective or cost function, e.g., power consumption).
4. Use probabilistic transition rules, not deterministic rules.

Parent Selection:
Select a parent pool at random from the population. Choose the two most fit in that pool for mating.

Mating: Crossover
Chromosomes from the two parents split and recombine at random locations along a gene backbone to create chromosomes for a child.



Gene Mutations:
A random event where a gene within a chromosome switches state: recessive $\leftrightarrow$ dominant.

Class Interface: Genes

For Its Implementation: Code Has Been Uploaded to eCampus

A. D. Freed

```
1  class Gene(object):
2
3      # base gene type: 0 is recessive, 1 is dominant
4      haploid = (0, 1)
5
6      def __init__(self, geneExpression=None):
7
8      def pop(self):
9
10     def push(self, geneExpression):
11
12     def copy(self):
13
14     def isEqualTo(self, gene):
15
16     def toString(self):
17
18     # the probability of mutation must lie within (0,1)
19     def mutate(self, probabilityOfMutation):
```

Class Interface: Chromosomes

Crossover Is a Function Requiring 2 Chromosomes; It Is Not a Method of 1 Chromosome


```
1  class Chromosome(object):
2      recessive = Gene(0)
3      dominant = Gene(1)
4      def __init__(self, minValue, maxValue, nbrSigFigs):
5      def _binaryToGray(self):
6      def _grayToBinary(self):
7      def _binaryToInteger(self):
8      def _integerToBinary(self, i):
9      def _integerToPhenotype(self, i):
10     def _phenotypeToInteger(self, p):
11     def pop(self, atLocation):
12     def push(self, gene, toLocation):
13     def copy(self):
14     def isEqualTo(self, c):
15     def toString(self):
16     def mutate(self, probOfMutation):
17     def decode(self):
18     def encode(self, phenotype):
19     def genes(self):
20
21 def Crossover(parentA, parentB, probOfMutation, probOfCrossover):
```

Class Interface: Genome

Crossover Becomes a Method for Genome

A. D. Freed

```
1  class Genome(object):
2
3      def __init__(self, minValues, maxValues, nbrSigFigs):
4
5      def pop(self, atLocation):
6      def push(self, chromosome, toLocation):
7      def copy(self):
8      def isEqualTo(self, g):
9      def toString(self):
10
11     def mutate(self, probOfMutation):
12     def decode(self):
13     def encode(self, phenotypes):
14     def genes(self):
15     def chromosomes(self):
16
17     def crossover(self, parentB, probOfMutation, probOfCrossover):
```

Class Interface: Creatures

A Population of Creatures Makes a Colony

A. D. Freed

```
1  class Creature(object):
2      genome = None
3      def __init__(self, varyPar, fixedPar, minVaryPar, maxVaryPar,
4                   nbrSigFigs, probOfMutation, probOfCrossover):
5      def _assignParameters(self, assigned):
6
7      # these are the three constructors for this class
8      def procreate(self):
9      def alien(self, alienPar):
10     def conceive(self, parentA, parentB):
11
12     # the following are its methods
13     def copy(self):
14     def isEqualTo(self, c):
15     def toString(self):
16     def genes(self):
17     def chromosomes(self):
18     def getParameters(self):
19     def getFitness(self):
20     def setFitness(self, fitness):
```

Class Interface: GenAlg

Evolve a Colony of Creatures over Generations

A. D. Freed

```
1  class GenAlg(object):
2      alphabet = 2      # haploid genes have two expressions: dominant & re
3
4      # the constructor
5      def __init__(self, species, nbrSigFigs, probOfMutations,
6                   probOfCrossovers, probOfImmigrants, namePar,
7                   varyPar, fixedPar, alienPar, minVaryPar, maxVaryPar):
8
9      #  methods to advance to the next generation
10     def _tournamentPlay(self):
11     def _mate(self):
12     def _evaluate(self, creature):
13     def advanceToNextGeneration(self):
14     def generationsToConvergence(self):
15
16     # methods used to create a report
17     def bestCreatureParam(self):
18     def bestCreatureFitness(self):
19     def population(self):
20     def atGeneration(self):
21     def report(self):
```

Licensed Under GNU General Public License


```
1  Module genAlg.py provides a genetic algorithm for optimization.
2
3  Copyright (c) 2018 Alan D. Freed
4
5  This program is free software: you can redistribute it and/or modify
6  it under the terms of the GNU General Public License as published by
7  the Free Software Foundation, either version 3 of the License, or
8  (at your option) any later version.
9
10 This program is distributed in the hope that it will be useful,
11 but WITHOUT ANY WARRANTY; without even the implied warranty of
12 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 GNU General Public License for more details.
14
15 You should have received a copy of the GNU General Public License
16 along with this program. If not, see <https://www.gnu.org/licenses/>.
```

Class Interface: Species

How Users Use this Genetic Algorithm

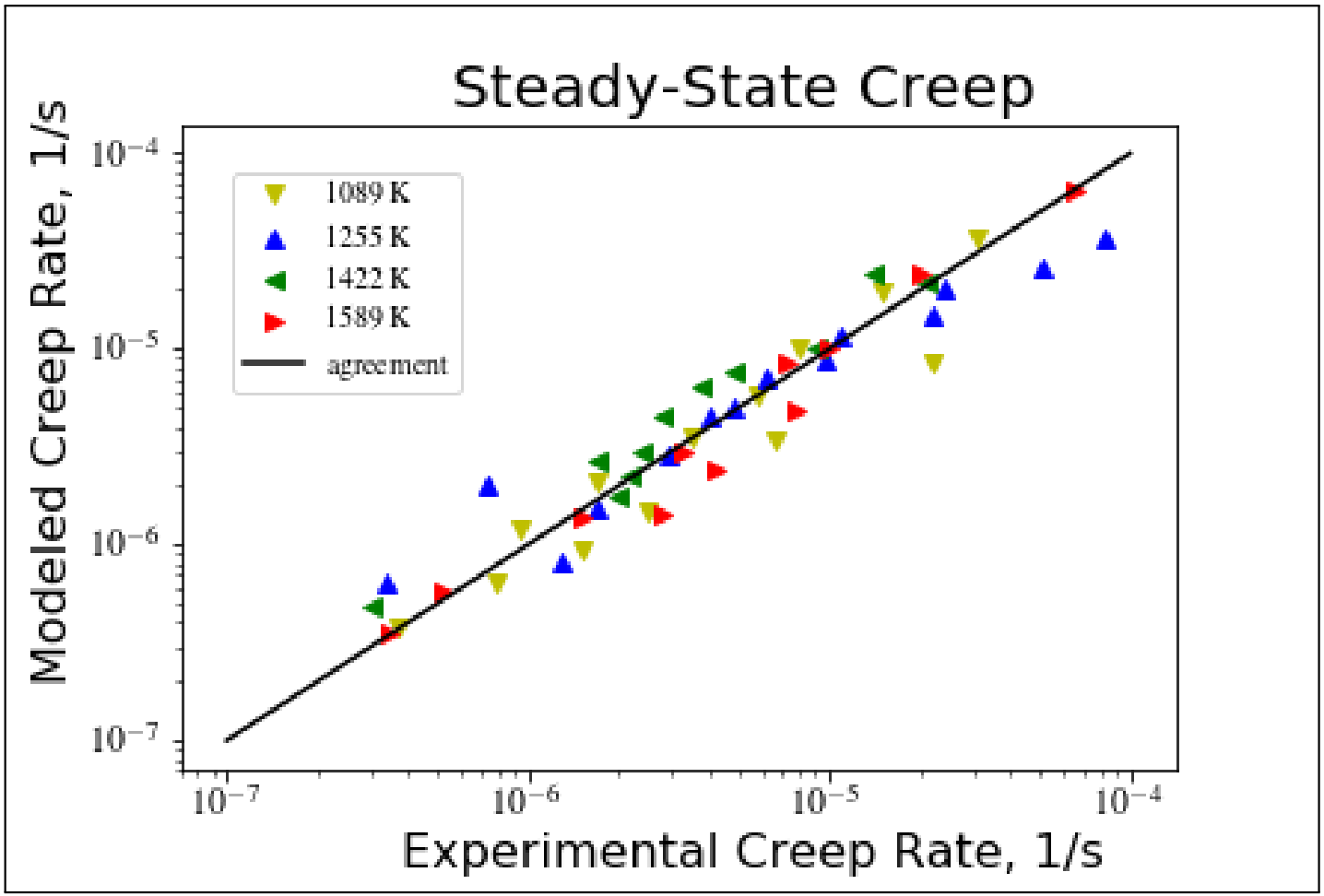
A. D. Freed

```
1  class Specie(object):
2
3      # Constructor: must be called via super by all child objects.
4      def __init__(self, expTypes, ptsInExp, ctrlsInExp, respsInExp):
5
6      # Create data structure for the experimental control variables.
7      def newExpCtrlData(self):
8
9      # Assign structure for the experimental control variables.
10     def assignExpCtrlData(self, controlData):
11
12     # Create data structure for entering exp. response variables.
13     def newExpRespData(self):
14
15     # Assign data structure for the experimental response variables.
16     def assignExpRespData(self, responseData):
17
18     # The model's interface to the genetic algorithm.
19     # Must be called via super by all child objecets.
20     def model(self, parameters):
```

Example: Steady State Creep of Chromium

Running testGenAlg.py from Software Distribution

A. D. Freed



Results of an Optimization Run Using Genetic Algorithm

Example: Steady State Creep of Chromium

Report Generated by Genetic Algorithm
Statistics from Each Generation Are Reported

A. D. Freed

```
1  Statistics for generation 47 with a population size of 272.
2  Optimal value and population statistics for fitness:
3  optimal value    8.5310E+01
4  arithmatic mean  4.1557E+01
5  median           4.7978E+01
6  std deviation    1.6538E+01
7  skewness        -1.0152E+00
8  excess kurtosis -2.5463E+00
9
10 ... Like data are displayed for each parameter determined.
11
12 Genetic genome for the most fit creature:
13 0: 00010000110110001000110010111111010000011101001
14 1: 11010000011000011010000
15 2: 00100101111011100110011
16 3: 00111010011100111011000l
17 4: 01010011100010111010101O
18 5: 11000110110110011111111l
19 Parameters assigned fixed values:
20 Tm:  2.1730E+03
```

Thank you! Questions?

What the Milky Way looks like from outside the galaxy. A bar galaxy. We are in the orange dot below the center.

A. D. Freed