Optimization: Performance Surface

- Problem: Minimize or maximize a performance criterion in terms of the variables
- Can be formulated as a mathematical expression as a surface against the space of the variables
Special Case: Unimodal Performance Surface

• There is a single (global) minimum or maximum, for example, a quadratic surface

• The goal is to find the minimum $w^*$ that realizes minimum performance value $J_{\text{min}}$

• One may be able to evaluate $J(w)$, $\nabla J(w)$
Steepest Descent Optimization

- At each point evaluate the slope (gradient) $\nabla J(w)$ of the performance surface $J(w)$ with respect to parameters $w$
- Move in the (negative) direction of the gradient towards the bottom of the performance curve
- $w_{k+1} = w_k - \mu \nabla J(w_k)$
Unimodal Performance Surface
Steepest Descent Trajectory

- At each step the descent is in the maximum possible rate.
- Depending on the value $\mu$, one can approach the minimum fast.
- Depending on the value $\mu$, there may be oscillations around $J_{\text{min}}$.
- An estimate for $\nabla J(w_k)$ can be substituted.
- Well-known example: LMS algorithm.
LMS Algorithm

- Employs a digital filter with a series of delay elements \((z^{-1})\) and variable weights \(b_i, i = 0, 1, \ldots, M\)
- There is a desired response \(d(n) = d_k\)
- The error is \(e(n) = d(n) - y(n) = e_k\)
- The goal is to minimize mean squared error \(E[e^2(n)]\)
- Approximating the true gradient with its stochastic value
  \[ w_{k+1} = w_k + \mu e_k x_k \]
Rosenbrock Function

- Nonconvex function used as a test problem for optimization algorithms
- \( f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \)
- **Global minimum at** \( (x, y) = (1, 1) \) **where** \( f(1, 1) = 0 \)
Steepest Descent with Rosenbrock Function

- The Rosenbrock function has a narrow curved valley which contains the minimum.
- The bottom of the valley is very flat.
- Because of the curved flat valley the optimization is zig-zagging slowly with small step sizes towards the minimum.
Making Steepest Descent Converge Faster

Steepest descent type algorithms that take the second derivative into account

- Newton-Raphson
- Polack-Ribiere
- Davidson-Fletcher-Powell
- Broydon-Fletcher-Goldfarb-Shanno
- Others
Nonunimodal Performance Surface
Gradient Ascent

\[ f(x, y) = \left( \frac{x^2}{2} - \frac{y^2}{4} + 3 \right) \cos (2x + 1 - e^y) \]

- Steepest descent is not suitable for nonunimodal performance surfaces
- It gets stuck at a local minimum/maximum
How to Escape Local Minima?

Initial position of the ball

Steepest descent gets stuck here

Need algorithm that will attempt to climb out of local minima

Algorithm should converge to global minimum after a sufficiently long run
Simulated Annealing: Motivation

• Annealing in metals
• Heat the solid state metal to a high temperature
• Cool it down very slowly according to a specific schedule.
• If the heating temperature is sufficiently high to ensure random state and the cooling process is slow enough to ensure thermal equilibrium, then the atoms will place themselves in a pattern that corresponds to the global energy minimum of a perfect crystal (Metropolis et al., 1953)
Simulated Annealing

Step 1: Initialize – Start with a random initial placement. Initialize a very high “temperature.”

Step 2: Move – Perturb the placement through a defined move.

Step 3: Calculate score – calculate the change in the score due to the move made.

Step 4: Choose – Depending on the change in score, accept or reject the move. The probability of acceptance depends on the current “temperature.”

Step 5: Update and repeat – Update the temperature value by lowering the temperature. Go back to Step 2.

The process is carried out until “Freezing Point” is reached.
Cooling Schedule
SA Flowchart

1. **Initialisation**
2. Metropolis simulation with fixed temperature $T$
3. Generate new solution
4. Adjust the solution
5. Evaluate cost function
6. **Improvement**
   - Yes: Accept new solution
   - No: Accept new solution with a probability
7. **Check for equilibrium**
   - Yes: Stop criteria at outer loop
   - No: Cooling temperature $T$
8. **Return optimal solution**
Convergence of SA

- Move accepted with probability $e^{-\Delta/v_{\text{temp}}}$

- At Init_Temp: Unconditional Acceptance

- At Final_Temp: HILL CLIMBING

Cost Function, $C$

Number of Iterations
Simple Example

The Traveling Salesman Problem

Find a tour of a given set of cities so that

- Each city is visited only once
- The total distance traveled is minimized
Simulated Annealing Solution of the Traveling Salesman Problem
Simulated Annealing Solution of the Traveling Salesman Problem
Simulated Annealing Solution of the Traveling Salesman Problem
SA Properties

- SA is a general solution method that is easily applicable to a large number of problems
- "Tuning" of the parameters is relatively easy
- Generally the quality of the results of SA is good, although it can take a lot of time
- Results are generally not reproducible: another run can give a different result
- SA can leave an optimal solution and not find it again (so remember the best solution found so far)
- Proven to find the optimum under certain conditions; one of these conditions is that you must run forever
SA Applications

- Computer-aided circuit design (placement, routing, etc) [IBM, Berkeley]
- Signal processing
- Boltzmann machines (AI)
- Operations research
- Econometrics
- Biology: Protein structure calculations, Gene clustering, etc.
Genetic Algorithms

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970s)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems
- Provide efficient, effective techniques for optimization and machine learning applications
- Widely used today in business, science, and engineering
Components of a GA

A problem to solve, and ...

• **Encoding technique** (gene, chromosome)
• **Initialization procedure** (creation)
• **Evaluation function** (environment)
• **Selection of parents** (reproduction)
• **Genetic operators** (mutation, recombination)
• **Parameter settings** (practice and art)
GA Cycle of Reproduction

- Reproduction
  - children
  - Parents
  - Population
    - Evaluated children
    - Deleted members
    - Discard
  - Evaluation
    - Modified children
  - Modification
Population

Chromosomes could be

- Bit strings  
  (0101 ... 1100)
- Real numbers  
  (43.2 -33.1 ... 0.0 89.2)
- Permutations of element  
  (E11 E3 E7 ... E1 E15)
- Lists of rules  
  (R1 R2 R3 ... R22 R23)
- Program elements  
  (genetic programming)
- Any data structure
Parents are selected at random with selection chances biased in relation to chromosome evaluations.
Chromosome Modification

- Modifications are stochastically triggered
- Operator types are
  - Crossover (recombination)
  - Mutation
Crossover: Recombination

P1  (0 1 1 0 1 0 0 0)  (0 1 0 0 1 0 0 0)  C1
P2  (1 1 0 1 1 0 1 0)  (1 1 1 1 1 0 1 0)  C2

- Crossover is a critical feature of genetic algorithms
  - Greatly accelerates search early in evolution of a population
  - Leads to effective combination of schemata (subsolutions on different chromosomes)
Mutation: Local Modification

Before: (0 0 1 1 0 1 1 0)
After: (1 1 1 1 0 0 1 1 0)

Before: (1.38 -69.4 326.44 0.1)
After: (1.38 -67.5 326.44 0.1)
Evaluation

- The evaluator decodes a chromosome and assigns it a fitness measure.
- The evaluator is the only link between a classical GA and the problem it is solving.
Deletion

- **Generational GA:** Entire populations replaced with each iteration
- **Steady-State GA:** A few members replaced each generation
Simple Example

The Traveling Salesman Problem

Representation is an ordered list of city numbers

1) London   3) Dunedin   5) Beijing   7) Tokyo
2) Venice   4) Singapore   6) Phoenix   8) Victoria

CityList1  (3 5 7 2 1 6 4 8)
CityList2  (2 5 7 6 8 1 3 4)
Crossover Example

Crossover combines inversion and recombination:

Parent1 \( (3 \ 5 \ 7 \ 2 \ 1 \ 6 \ 4 \ 8) \)
Parent2 \( (2 \ 5 \ 7 \ 6 \ 8 \ 1 \ 3 \ 4) \)
Child \( (5 \ 8 \ 7 \ 2 \ 1 \ 6 \ 3 \ 4) \)

This operator is called the Order1 crossover.
Mutation Example

Mutation involves reordering of the list

Before: \[(5 \ 8 \ 7 \ 2 \ 1 \ 6 \ 3 \ 4)\]

After: \[(5 \ 8 \ 6 \ 2 \ 1 \ 7 \ 3 \ 4)\]
Traveling Salesperson Problem

“A salesman must travel to N number of cities, visiting each city once before returning to the start city. The goal is for the salesman to travel the shortest distance. Note: The red line connecting the dots when the genetic algorithm is run represents the salesman’s path, and the blue dot represents a city.”
Traveling Salesperson Problem:

“A salesman must travel to N number of cities, visiting each city once before returning to the start city. The goal is for the salesman to travel the shortest distance. Note: The red line connecting the dots when the genetic algorithm is run represents the salesman’s path, and the blue dot represents a city.”

Working... Generation: 1028

Tip: Play around with different crossover and mutation settings to increase the rate of evolution.

Contact Us: support@rubicite.com
Traveling Salesperson Problem:
“A salesman must travel to N number of cities, visiting each city once before returning to the start city. The goal is for the salesman to travel the shortest distance. Note: The red line connecting the dots when the genetic algorithm is run represents the salesman’s path, and the blue dot represents a city.”

Working... Generation: 2018

Tip: Play around with different crossover and mutation settings to increase the rate of evolution.
Traveling Salesperson Problem:

“A salesman must travel to N number of cities, visiting each city once before returning to the start city. The goal is for the salesman to travel the shortest distance. Note: The red line connecting the dots when the genetic algorithm is run represents the salesman’s path, and the blue dot represents a city.”

Working... Generation: 12518

Tip: Play around with different crossover and mutation settings to increase the rate of evolution.
Traveling Salesperson Problem

A salesman must travel to N number of cities, visiting each city once before returning to the start city. The goal is for the salesman to travel the shortest distance. Note: the red line connecting the dots when the genetic algorithm is run represents the salesman’s path, and the blue dot represents a city.

Reports:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Fitness</th>
<th>Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>13</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>16</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>17</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>18</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>19</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>21</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>22</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>23</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>24</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>25</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>26</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>27</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>28</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>29</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>31</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>32</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>33</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>34</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>35</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>36</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>37</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>38</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>39</td>
<td>5476.19</td>
<td>100</td>
</tr>
<tr>
<td>40</td>
<td>5476.19</td>
<td>100</td>
</tr>
</tbody>
</table>

Tip: Play around with different crossover and mutation settings to increase the rate of evolution.

Contact Us: support@rubcite.com
GA Applications

- Automated design, including research on composite material design and design of automotive components for crashworthiness, weight savings, and other characteristics.
- Automated design of mechatronic systems using bond graphs and genetic programming
- Automated design of industrial equipment using catalogs of exemplar lever patterns.
- Automated design of sophisticated trading systems in the financial sector.
- Building phylogenetic trees
- Chemical kinetics (gas and solid phases)
- Configuration applications, particularly physics applications of optimal molecule configurations
- Container loading optimization
- Code-breaking, using the GA to search large solution spaces of ciphers for the one correct decryption
- Topology
- Electronic circuit design
- File allocation for a distributed system
- Game Theory: Equilibrium Resolution.
- Gene expression profiling analysis
- Linguistic analysis, including Grammar induction and other aspects of Natural language processing (NLP)
- Mobile communications infrastructure optimization
- Molecular structure optimization (Chemistry)
- Multiple criteria production scheduling
- Multiple population topologies and interchange methodologies
GA Applications

- Mutation testing
- Neural networks
- Optimization of data compression systems, for example using wavelets
- Music
- Protein folding and protein/ligand docking
- Representing rational agents in economic models such as the cobweb model
- Bioinformatics: RNA structure prediction
- Bioinformatics: [Multiple Sequence Alignment]
- Bioinformatics: Multiple sequence alignment
- Scheduling applications, including job-shop scheduling. Selection of optimal mathematical model to describe biological systems.
- Software engineering
- Solving the machine-component grouping problem required for cellular manufacturing systems allocation and strategies.
- Timetabling problems, such as designing a non-conflicting class timetable for a large university
- Training artificial neural networks when pre-classified training examples are not readily obtainable
- Traveling Salesman Problem
- Finding hardware bugs
- Wireless Sensor/Ad-hoc Networks
- Data Center/Server Farm
Related Techniques

- **Tabu Search**: SA-like. Search by testing *multiple* mutations with a tabu list (unacceptable solutions). Choose the minimum energy solution.

- **Ant Colony Optimization**: To solve a path problem, first find a *solution*, then search the space in parallel to drop longer segments to get to the shortest path.

- **Bacteriologic Algorithms**: Based on GA. To find solutions good for a *population*, not a single best solution.

- **Memetic Algorithms**: Based on the concept of Universal Darwinism due to Dawkins which says principles of evolution are applicable not only to genes but all complex systems that exhibit inheritance, variation, and selection. Memes, unlike genes, can adapt themselves. A memetic algorithm performs local search during the evolutionary cycle.
Related Techniques

- **Cultural Algorithm**: Similar to GA, with an additional knowledge component, called the belief space.

- **Cross-Entropy Method**: Generate candidate solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization to generate better samples in the next iteration.

- **Others**: Evolution Strategies, Evolutionary Programming, Extremal Optimization, Gaussian Adaptation, Genetic Programming, Interactive Evolutionary Algorithms, Harmony Search, Reactive Search Optimization
Hidden Markov Models

- Markov model whose states are observable only through probabilistic observations
- Model observation sequences with short-term stationary characteristics, occasional changes
- Originally used in speech recognition, later applied to biological sequence analysis
1. Given an observation sequence and the model, what is the probability that the observation sequence was generated by the model?

   Forward-Backward Algorithm

2. Given an observation sequence how to choose a state sequence optimum in some sense?

   Viterbi Algorithm

3. How adjust model parameters to maximize probability of observations given the model?

   Baum-Welch Algorithm, etc.
Overview

Profile hidden Markov models (profile HMMs) can be used to do sensitive database searching using statistical descriptions of a sequence family’s consensus. HMMER is a freely distributable implementation of profile HMM software for protein sequence analysis.

The current version is HMMER 2.3.2 (3 Oct 2003), containing minor bugfixes and updates for the May 2003 release of HMMER 2.3.

HMMER3 beta test in progress

HMMER 3 is in a public beta test phase. Some of the major improvements include a heuristic search algorithm that makes HMMER about as fast as BLAST (while retaining the sensitivity of HMM-based approaches), combined with new statistical theory for profile HMM log-likelihood scores (Eddy, 2008) that allows us to use more powerful full likelihood approaches (summing over alignments, rather than having to score only the optimal one) than HMMER has used in the past. The beta test code is showing large increases in both speed and sensitivity.

HMMER3 source code and Linux binaries are available for download as a [tarball on our FTP site]. You may also view the release notes for the current test code, or the rudimentary User’s Guide.

HMMER3 is now reasonably stable. I expect the beta test to last a couple of months, during which we’ll be documenting, ironing out some remaining issues behind the scenes, and fixing any bugs that get smoked out. I’m hoping for public 3.0 release in summer 2009.
A hidden Markov model that finds genes in *E.coli* DNA

Anders Krogh, I.Saira Mian\(^1\) and David Haussler\(^2\)*
Nordita, Blegdamsvæj 17, DK-2100 Copenhagen, Denmark, \(^1\)Sinsheimer Laboratories, University of California, Santa Cruz, CA 95064 and \(^2\)Computer and Information Sciences, University of California, Santa Cruz, CA 95064, USA

Received June 21, 1994; Revised and Accepted September 26, 1994

**ABSTRACT**

A hidden Markov model (HMM) has been developed to find protein coding genes in *E.coli* DNA using *E.coli* genome DNA sequence from the EcoSeq\(\text{6}\) database maintained by Kenn Rudd. This HMM includes states that model the codons and their frequencies in *E.coli* genes, as well as the patterns found in the intergenic region, including repetitive extragenic palindromic sequences and the Shine–Dalgarno motif. To account for potential sequencing errors and or frameshifts in raw genomic DNA sequence, it allows for the (very unlikely) possibility of insertions and deletions of individual nucleotides within a codon. The parameters of the HMM are estimated using approximately one million nucleotides of annotated DNA in EcoSeq\(\text{6}\) and the model tested on a disjoint set of contigs containing about 325,000 nucleotides. The HMM finds the exact locations of about 80% of the known *E.coli* genes, and approximate locations for about 10%. It also finds several potentially new genes, and locates several places were insertion or deletion errors/and or frameshifts may be present in the contigs.

non-coding region (reviewed in [10]). Staden and McLachlan [11,3] proposed deviation from average codon usage as a way of determining the probability that the window is coding or not. Later, Gribkov et al. [12] used a similar measure as a part of their 'codon preference plot', but their measure did not require the knowledge of an average codon usage from other sources. Most other scoring methods are related to codon usage in some way [13,3]. Recently, neural networks [4,14,15,16] and Markov chains [17,18,19] have been used to analyze coding (and non-coding) regions. In particular, the program GeneMark [20] finds genes in *E.coli* DNA using a Markov model for the coding region related to the one discussed here, and a very simple Markov model for the non-coding regions. Whether looking for signals in the DNA or using window scoring, there remains the problem of combining all the scores and/or signals detected in a given contig to produce a coherent 'parse' into genes separated by intergenic regions. The output of this final parsing step could be a list of genes, each represented by its begin and end position within the contig. Snyder and Stormo have recently proposed an elegant dynamic programming method to accomplish this final step [21]. Other more linguistically motivated approaches to this kind of sequence parsing problem are described in [22,23,24,25].
Engineering the Future at UC Irvine