Advanced Algorithmics (6EAP)

Search and meta-heuristics

Jaak Vilo

2017 Fall
Search

• for what?
  – a solution
  – the (best possible (approximate?)) solution

• from where?
  – search space (all valid solutions or paths)

• under which conditions?
  – compute time, space, ...
  – constraints, ...
Objective function

• An optimal solution
  – what is the measure that we optimise?
    • Any solution (satisfiability /SAT/ problem)
      – does the task have a solution?
      – is there a solution with objective measure better than X?
    • Minimal/maximal cost solution
    • A winning move in a game
    • A (feasible) solution with smallest nr of constraint violations (e.g. course time scheduling)
Search space size?

• Linear (list, binary search, ...)
• Integer in [i,j]
• Real nr in [x,y)
• A point in high-dimensional space
• An assignment of variables (in SAT)
• A subset of a larger set
• Trees, Graphs
• ...

The Propositional Satisfiability Problem (SAT)

Simple SAT instance (in CNF):

\[(a \lor b) \land (\neg a \lor \neg b)\]

\[\sim\] satisfiable, two models:

\[a = \text{true}, \ b = \text{false}\]
\[a = \text{false}, \ b = \text{true}\]
Solution search space: lattice for 4 variables

O($2^n$)
Tic-Tac-Toe

It is straightforward to write a computer program to play Tic-tac-toe perfectly, to enumerate the 765 essentially different positions (the state space complexity), or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space.
TSP, nearest neighbour search

An Instance of the Traveling Salesman Problem

Search Space

abcdea
375

abceda
425
An Instance of the Traveling Salesman Problem

Cost of Nearest Neighbor Path, \( AEDBCA = 550 \)
1 Introduction

Figure 1.2: Global and local optima of a two-dimensional function.
Issues:
Constraints

• Time, space...
  – if optimal cannot be found, approximate

• All kinds of secondary characteristics

• Constraints
  – sometimes finding even a point in the valid search space is hard
# Types of games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Perfect Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chess, checkers, go, othello</td>
<td>Backgammon, monopoly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Imperfect Information</th>
<th>Deterministic</th>
<th>Imperfect Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battleships, blind tictactoe</td>
<td>Bridge, poker, scrabble nuclear war</td>
<td></td>
</tr>
</tbody>
</table>
# Test functions for single-objective optimization problems

<table>
<thead>
<tr>
<th>Name</th>
<th>Plot</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rastrigin function</strong></td>
<td><img src="https://en.wikipedia.org/wiki/Test_functions_for_optimization" alt="Rastrigin Function Plot" /></td>
<td>[f(x) = An + \sum_{i=1}^{n} \left[x_i^2 - A \cos(2\pi x_i)\right]] where: (A = 10)</td>
</tr>
</tbody>
</table>
| **Ackley's function** | ![Ackley's Function Plot](https://en.wikipedia.org/wiki/Test_functions_for_optimization) | \[f(x, y) = -20 \exp\left(-0.2 \sqrt{0.5 \left(x^2 + y^2\right)}\right)\
\quad - \exp\left(0.5 \left(\cos(2\pi x) + \cos(2\pi y)\right)\right) + e + 20\] |
| **Sphere function**   | ![Sphere Function Plot](https://en.wikipedia.org/wiki/Test_functions_for_optimization)   | \[f(x) = \sum_{i=1}^{n} x_i^2\]                                       |
| **Rosenbrock function**| ![Rosenbrock Function Plot](https://en.wikipedia.org/wiki/Test_functions_for_optimization) | \[f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]\] |
Virtual Library of Simulation Experiments: Test Functions and Datasets

Optimization Test Problems

GRIEWANK FUNCTION

\[ f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \]
An interesting constrained numerical optimization test case emerged recently; the problem (Keane, 1994) is to maximize a function:

\[ G^2(x) = \left| \sum_{i=1}^{n} \frac{\cos^4(x_i) - 2 \prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, \]

subject to

\[ \prod_{i=1}^{n} x_i \geq 0.75 \quad \text{and} \quad \sum_{i=1}^{n} x_i \leq 7.5n \quad \text{and bounds} \quad 0 \leq x_i \leq 10 \quad \text{for} \quad 1 \leq i \leq n. \]

Function \( G^2 \) is nonlinear and its global maximum is unknown, lying somewhere near the origin. The problem has one nonlinear constraint and one linear constraint; the latter one is inactive around the origin and will be forgotten in the following.

\[ G^2(x) = (\sum \cos^4(x_i) - 2 \prod \cos^2(x_i))/\sqrt{\sum i x_i^2}, \]

where \( 0 \leq x_i \leq 10 \) and

\[ \prod x_i \geq 0.75 \]
The graph of function $G2$ for $n = 2$. Infeasible solutions were as
In numerical analysis, **Newton's method** (also known as the **Newton–Raphson method**), named after **Isaac Newton** and **Joseph Raphson**, is a method for finding successively better approximations to the **roots** (or zeroes) of a **real-valued function**.

\[ x : f(x) = 0. \]

The Newton-Raphson method in one variable is implemented as follows:

Given a function \( f \) defined over the reals \( x \), and its **derivative** \( f' \), we begin with a first guess \( x_0 \) for a root of the function \( f \). Provided the function satisfies all the assumptions made in the derivation of the formula, a better approximation \( x_1 \) is

\[ x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \]

Geometrically, \((x_1, 0)\) is the intersection with the \( x \)-axis of a line **tangent** to \( f \) at \((x_0, f(x_0))\).

The process is repeated as

\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \]

until a sufficiently accurate value is reached.
Examples

• Greedy
• A* search
• Monte Carlo, Grid
• Local search algorithms - Hill-climbing, beam, ...
• Simulated annealing search
• Genetic algorithms (GA, GP)
• Differential Evolutions (DE)
• Particle Swarm Optimisation (PSO)
• Ant colony optimisation (ACO)
Classes of Search Techniques

- Calculus-based techniques
- Indirect methods
- Direct methods
  - Fibonacci
  - Newton
- Guided random search techniques
  - Evolutionary algorithms
  - Simulated annealing
- Enumerative techniques
  - Dynamic programming
- Evolutionary strategies
  - Genetic algorithms
    - Parallel
      - Centralized
      - Distributed
    - Sequential
      - Steady-state
      - Generational
Greedy

- Always choose the next seemingly best step
Set Cover

Greedy Approximation Algorithm

– polynomial-time \( \rho(n) \)-approximation algorithm
  
  • \( \rho(n) \) is a logarithmic function of set size
  
  • \( n \) – size of the largest set...

  
  • \( | C(\text{greedy}) | < \ln(n) \times | C(\text{optimal}) | \)
Set Cover Problem

Instance \((X, \mathcal{F})\):

- finite set \(X\) (e.g. of points)
- family \(\mathcal{F}\) of subsets of \(X\)

\[ X = \bigcup_{S \in \mathcal{F}} S \]

Problem: Find a minimum-sized subset \(C \subseteq \mathcal{F}\) whose members cover all of \(X\):

\[ X = \bigcup_{S \in C} S \]
Greedy Set Covering Algorithm

**Greedy-Set-Cover**($X$, $F$)

1. $U \leftarrow X$
2. $C \leftarrow \emptyset$
3. while $U \neq \emptyset$
4. do select an $S \in F$ that maximizes $|S \cap U|$
5. $U \leftarrow U - S$
6. $C \leftarrow C \cup \{S\}$
7. return $C$

Greedy: select set that covers the most uncovered elements

source: 91.503 textbook Cormen et al.
Set Cover

**Theorem:** GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm for

$$\rho(n) = H(\max \{| S | : S \in \mathcal{F} \})$$

**Proof:**

$d$th harmonic number $H_d = \sum_{i=1}^{d} \frac{1}{i} = H(d)$  \(H(0) = 0\)

Algorithm runs in time polynomial in $n$.

$S_i = \text{ith subset selected}$  \(selecting S_i \text{ costs 1}\)

$c_x = \text{cost of element } x \in X$  \(\text{paid only when } x \text{ is covered for the first time}\)

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|}$$

Assume $x$ is covered for the first time by $S_i$

(spread cost evenly across all elements covered for first time by $S_i$)

Number of elements covered for first time by $S_i$
The term harmonic number has multiple meanings. For other meanings, see harmonic number (disambiguation).

In mathematics, the \( n \)-th harmonic number is the sum of the reciprocals of the first \( n \) natural numbers:

\[
H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}.
\]

This also equals \( n \) times the inverse of the harmonic mean of these natural numbers.

Harmonic numbers were studied in antiquity and are important in various branches of number theory. They are sometimes loosely termed harmonic series, are closely related to the Riemann zeta function, and appear in various expressions for various special functions.

When the value of a large quantity of items has a Zipf's law distribution, the total value of the \( n \) most-valuable items is the \( n \)-th harmonic number. This leads to a variety of surprising conclusions in the Long Tail and the theory of network value.

\[
\begin{align*}
\text{The harmonic number } H_{n,1} \text{ with } n = \lfloor x \rfloor \text{ (red line) with its asymptotic limit } \gamma + \ln[x] \text{ (blue line).}
\end{align*}
\]
Proof of approximation

• http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/wan-ba-notes.pdf
Set Cover (proof continued)

**Theorem:** GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm for

$$\rho(n) = H(\max \{|S| : S \in \mathcal{F}\})$$

**Proof:** (continued)

Let $C^*$ be an optimal cover.

$C$ be cover from GREEDY - SET - COVER.

Cost assigned to optimal cover:

$$|C| = \sum_{x \in X} c_x$$

Each $x$ is in $\geq 1$ $S$ in $C^*$

$$\sum_{S \in C^*} \left( \sum_{x \in S} c_x \right) \geq \sum_{x \in X} c_x$$

$$|C| \leq \sum_{S \in C^*} \left( \sum_{x \in S} c_x \right)$$

1 unit is charged at each stage of algorithm.
Set Cover (proof continued)

**Theorem:** GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm for

$$\rho(n) = H(\max\{|S| : S \in \mathcal{F}\})$$

**Proof:** (continued)

How does this relate to harmonic numbers??

$d$th harmonic number $H_d = \sum_{i=1}^{d} \frac{1}{i} = H(d)$

We’ll show that:

$$\sum_{x \in S} c_x \leq H(|S|)$$

for any set $S \in \mathcal{F}$

And then conclude that:

$$|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| H(\max\{|S| : S \in \mathcal{F}\})$$
Local Search
Local search algorithms

• In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

• State space = set of "complete" configurations
• Find configuration satisfying constraints, e.g., n-queens

• In such cases, we can use local search algorithms
• keep a single "current" state, try to improve it
Example: $n$-queens

• Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

• may get stuck...
Problems

• Problem: Cycles
  – Memorize
  – Tabu search

• How to transfer valleys with bad choices only...
Tree/Graph search

- order defined by picking a node for expansion
- BFS, DFS
- Random, Best First, Beam Search, pruning...
  - Best – an evaluation function
• Idea: use an **evaluation function** \( f(n) \) for each node
  – estimate of "desirability"
  – Expand most desirable unexpanded node

• **Implementation:**
  Order the nodes in fringe in decreasing order of desirability
  Priority queue

• **Special cases:**
  – greedy best-first search \( f(n) = h(n) \) **heuristic**, e.g. estimate to goal
  – A* search
A*

- $f(n) = g(n) + h(n)$
  - $g(n)$ – path covered so far in graph
  - $h(n)$ – estimated distance from $n$ to goal
Admissible heuristics

• A heuristic $h(n)$ is **admissible** if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$.

• An admissible heuristic never overestimates the cost to reach the goal, i.e., it is **optimistic**

• Example: $h_{SLD}(n)$ (never overestimates the actual road distance) (SLD – shortest linear distance)

• **Theorem**: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal
Optimality of A* (proof)

• Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.

\[
\begin{align*}
\cdot f(G_2) &= g(G_2) \quad \text{since } h(G_2) = 0 \\
\cdot g(G_2) &> g(G) \quad \text{since } G_2 \text{ is suboptimal} \\
\cdot f(G) &= g(G) \quad \text{since } h(G) = 0 \\
\cdot f(G_2) &> f(G) \quad \text{from above}
\end{align*}
\]
Optimality of A* (proof)

• Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.

• $f(G_2) = g(G_2)$ since $h(G_2) = 0$
• $g(G_2) > g(G)$ since $G_2$ is suboptimal
• $f(G) = g(G)$ since $h(G) = 0$
• $f(G_2) > f(G)$ from above
• Hence $f(G_2) > f(n)$, and A* will never select $G_2$ for expansion
A* path-finder

Better: http://www.youtube.com/watch?v=19h1g22hby8
Graph

• A (virtual) graph/search space
  – valid states of Fifteen-game
  – Rubik’s cube
Solve

• Which move takes us closer to the solution?
• Estimate the goodness of the state
Admissible heuristic for A*

• How many are misplaced? (7)

• How far have they been misplaced? Sum of theoretical shortest paths to the correct place

• A* search towards a final goal
Search Methods

• **Types of search methods:**
  • systematic $\leftrightarrow$ local search
  • deterministic $\leftrightarrow$ stochastic
  • sequential $\leftrightarrow$ parallel
Local Search (LS) Algorithms

• **search space** $S$
  (SAT: set of all complete truth assignments to propositional variables)

• **solution set** $S' \subseteq S$
  (SAT: models of given formula)

• **neighborhood relation** $N \subseteq S \times S$
  (SAT: neighboring variable assignments differ in the truth value of exactly one variable)

• **evaluation function** $g : S \rightarrow \mathbb{R}^+$
  (SAT: number of clauses unsatisfied under given assignment)
Local Search:

- start from initial position
- iteratively move from current position to neighboring position
- use evaluation function for guidance

Two main classes:

- local search on partial solutions
- local search on complete solutions
local search on partial solutions
Local search for partial solutions

- Order the variables in some order.
- Span a tree such that at each level a given value is assigned a value.
- Perform a depth-first search.
- But, use heuristics to guide the search. Choose the best child according to some heuristics. *(DFS with node ordering)*
DFS

• Once a solution has been found (with the first dive into the tree) we can continue search the tree with DFS and backtracking.
Construction Heuristics for partial solutions

• search space: space of partial solutions
• search steps: extend partial solutions with assignment for the next element
• solution elements are often ranked according to a greedy evaluation function
The Traveling Salesperson Problem (TSP)

• **TSP – optimization variant:**
  • For a given weighted graph $G = (V,E,w)$, find a Hamiltonian cycle in $G$ with minimal weight,
  • i.e., find the shortest round-trip visiting each vertex exactly once.

• **TSP – decision variant:**
  • For a given weighted graph $G = (V,E,w)$, decide whether a Hamiltonian cycle with minimal weight $\leq b$ exists in $G$. 
Nearest Neighbor heuristic for the TSP:

- at any city, choose the closest yet unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the $i$th step choose city $\pi(i + 1)$ to be the city $j$ that minimises $d(\pi(i), j); j \neq \pi(k), 1 \leq k \leq i$

- running time: $\mathcal{O}(n^2)$

- worst case performance:
  $$\frac{NN(x)}{OPT(x)} \leq 0.5([\log_2 n] + 1)$$

- other construction heuristics for TSP are available
Nearest neighbor tour through 532 US cities
TSP instance: shortest round trip through 532 US cities
Traveling salesman. Second, we consider the traveling salesman problem, which is formulated for a complete graph $G = (V, E)$ with a positive integer cost function $c : E \to \mathbb{Z}_+$. A tour in this graph is a Hamiltonian cycle and the problem is finding the tour, $A$, with minimum total cost, $c(A) = \sum_{uv \in A} c(uv)$. Let us first assume that the cost function satisfies the triangle inequality, $c(uw) \leq c(uv) + c(vw)$ for all $u, v, w \in V$. It can be shown that the problem of finding the shortest tour remains NP-complete even if we restrict it to weighted graphs that satisfy this inequality. We formulate an algorithm based on the observation that the cost of every tour is at least the cost of the minimum spanning tree, $C^* \geq c(T)$.

1. Construct the minimum spanning tree $T$ of $G$.
2. Return the preorder sequence of vertices in $T$. 
Figure 114: The solid minimum spanning tree, the dotted traversal using each edge of the tree twice, and the solid tour obtained by taking short-cuts.
My current best is 27486.199404966355 (nn gives 27766.484757657887)

All the best,

Polina
My best is 24839,308924381 (Jaak S)
My new best is 23474 (Oleg)
23297.72476804589
Probably some local minimum near Jaak Sarv's solution
A shortest-possible walking tour through the pubs of the UK

A shortest-possible walking tour through the pubs of the United Kingdom — that’s an advanced form of the mathematicians’ favorite, The Traveling Salesman Problem. William Cook and colleagues at the University of Waterloo tackled this nastily complex problem:

Nearly everyone in the UK knows by heart the best path to take them over to their favorite public house. But what about jotting down the shortest route to visit every pub in the country and return home safely? That is what we set out to do.

Using geographic coordinates of 24,727 pubs provided by Pubs Galore and measuring the distance between any two pubs as the length of the route produced by Google Maps, what is the shortest possible tour that visits all 24,727 and returns to the starting point? …

This is the problem we have solved. The optimal tour has length 45,495,239 meters. To be clear, our main result is that there simply does not exist any pub tour that is even one meter shorter (measuring the length using the distances we obtained from Google) than the one produced by our computation. It is the solution to a 24,727-city traveling salesman problem (TSP).

The UK Pubs tour is easily the largest such road-distance TSP that has been solved to date, having over 100 times more stops than any road-distance example solved previously by other research groups.

Here’s one low-resolution sliver of what is a much more detailed map of the tour:
A shortest-possible walking tour through the pubs of the UK
local search on complete solutions
Iterative Improvement (Greedy Search):

- initialize search at some point of search space
- in each step, move from the current search position to a neighboring position with better evaluation function value
Iterative Improvement for SAT

- **Initialization**: randomly chosen, complete truth assignment
- **Neighborhood**: variable assignments are neighbors iff they differ in truth value of one variable
- **Neighborhood size**: $O(n)$ where $n =$ number of variables
- **Evaluation function**: number of clauses unsatisfied under given assignment
Hill climbing

• Choose the neighbor with the largest improvement as the next state

\[
f\text{-value} = \text{evaluation}(\text{state})
\]

\[
\text{while } f\text{-value}(\text{state}) > f\text{-value}(\text{next-best}(\text{state}))
\]

\[
\text{state} := \text{next-best}(\text{state})
\]
Hill climbing

**function** Hill-Climbing(*problem*) **returns** a solution state

current ← Make-Node(Initial-State[*problem*])

**loop do**

next ← a highest-valued successor of current

if Value[next] < Value[current] then return current

current ← next

**end**
Problems with local search

Typical problems with local search (with hill climbing in particular)

• getting stuck in local optima
• being misguided by evaluation/objective function
**Stochastic Local Search**

- randomize initialization step
- randomize search steps such that suboptimal/worsening steps are allowed
- improved performance & robustness
- typically, degree of randomization controlled by noise parameter
Stochastic Local Search

Pros:
• for many combinatorial problems more efficient than systematic search
• easy to implement
• easy to parallelize

Cons:
• often incomplete (no guarantees for finding existing solutions)
• highly stochastic behavior
• often difficult to analyze theoretically/empirically
Simple SLS methods

• **Random Search (Blind Guessing):**
  • *In each step, randomly select one element of the search space.*

• **(Uninformed) RandomWalk:**
  • *In each step, randomly select one of the neighbouring positions of the search space and move there.*
Random restart hill climbing

\[ f\text{-value} = \text{evaluation}(\text{state}) \]
Randomized Iterative Improvement:

- initialize search at some point of search space search steps:
- with probability $p$, move from current search position to a randomly selected neighboring position
- otherwise, move from current search position to neighboring position with better evaluation function value.
- Has many variations of how to choose the randomly neighbor, and how many of them
- Example: Take 100 steps in one direction (Army mistake correction) – to escape from local optima.
Search space

- Problem: depending on initial state, can get stuck in local maxima
General iterative Algorithms

- general and “easy” to implement
- approximation algorithms
- must be told when to stop
- hill-climbing
- convergence
General iterative search

• Algorithm
  – Initialize parameters and data structures
  – construct initial solution(s)
  – Repeat
    • Repeat
      – Generate new solution(s)
      – Select solution(s)
    • Until time to adapt parameters
    • Update parameters
  – Until time to stop
• End
Iterative search

• Most popular algorithms of this class
  – Simulated Annealing
    • Probabilistic algorithm inspired by the annealing of metals
  – Tabu Search
    • Meta-heuristic which is a generalization of local search
  – Genetic Algorithms
    • Probabilistic algorithm inspired by evolutionary mechanisms
Simulated annealing

Combinatorial search technique inspired by the physical process of annealing [Kirkpatrick et al. 1983, Cerny 1985]

Outline

• Select a neighbor at random.
• If better than current state go there.
• Otherwise, go there with some probability.
• Probability goes down with time (similar to temperature cooling)
Search heuristics

$f(x,y)$
Search heuristics

\[ f(x, y) \]
Search heuristics

$f(x,y)$
Simulated Annealing

\[ f(x,y) \]

**IF** better **THEN** Accept
**ELSE** Accept with decreasing probability
Simulated annealing


• **Simulated annealing (SA)** is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

• The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

• By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter $T$ (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when $T$ is large, but increasingly "downhill" as $T$ goes to zero. The allowance for "uphill" moves potentially saves the method from becoming stuck at local optima—which are the bane of greedier methods.

• The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983,[1] and by Vlado Černý in 1985.[2] The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth in a paper by N. Metropolis et al. in 1953.[3]
Simulated annealing

Acceptance criterion

- Metropolis acceptance criterion
  - better solutions are always accepted
  - worse solutions are accepted with probability

$$e^{\Delta E/T} \sim \exp \left( \frac{g(s) - g(s')}{T} \right)$$

Annealing

- parameter $T$, called temperature, is slowly decreased
\[ \delta = -10 \]

Temperature cools down

\[ \text{Temperature cools down} \]
Acceptance probability

http://www.mi.fu-berlin.de/wiki/pub/Main/GunnarKlauP1winter0708/discMath_klau_script_meta.pdf

Gunnar Klau
Generic choices for annealing schedule

- initial temperature $T_0$
  (example: based on statistics of evaluation function)

- cooling schedule — how to change temperature over time
  (example: geometric cooling, $T_{n+1} = \alpha \cdot T_n, n = 0, 1, \ldots$)

- number of iterations at each temperature
  (example: multiple of the neighbourhood size)

- stopping criterion
  (example: no improved solution found for a number of temperature values)
Simulated Annealing


http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf
Pseudo code

function Simulated-Annealing(problem, schedule) returns solution state

current ← Make-Node(Initial-State[problem])

for t ← 1 to infinity

T ← schedule[t]  //  T goes downwards.

if T = 0 then return current

next ← Random-Successor(current)

ΔE ← f-Value[next] - f-Value[current]

if ΔE > 0 then current ← next

else current ← next with probability $e^{ΔE/T}$

end
s ← s0; e ← E(s)  // Initial state, energy.
sbest ← s; ebest ← e  // Initial "best" solution
k ← 0  // Energy evaluation count.
while k < kmax and e < emax  // While time left & not good enough:
    snew ← neighbour(s)  // Pick some neighbour.
enew ← E(snew)  // Compute its energy.
    if P(e, enew, temp(k/kmax)) > random() then  // Should we move to it?
        s ← snew; e ← enew  // Yes, change state.
        if enew > ebest then  // Is this a new best?
            sbest ← snew; ebest ← enew  // Save 'new neighbour' to 'best found'.
            k ← k + 1  // One more evaluation done
        return sbest  // Return the best solution found.
2-opt

Example application to the TSP [Johnson & McGeoch 1997]

baseline implementation:
• start with random initial solution
• use 2-exchange neighborhood
• simple annealing schedule;
→ relatively poor performance

improvements:
• look-up table for acceptance probabilities
• neighborhood pruning
• low-temperature starts
Diameter of the search graph

• Simulated annealing may be modeled as a random walk on a search graph, whose vertices are all possible states, and whose edges are the candidate moves. An essential requirement for the neighbour() function is that it must provide a sufficiently short path on this graph from the initial state to any state which may be the global optimum. (In other words, the diameter of the search graph must be small.) In the traveling salesman example above, for instance, the search space for \( n = 20 \) cities has \( n! = 2432902008176640000 \) (2.4 quintillion) states; yet the neighbour generator function that swaps two consecutive cities can get from any state (tour) to any other state in maximum \( n(n – 1) / 2 = 190 \) steps.
https://www.youtube.com/watch?v=W-aAjd8_bUc
Simulated Annealing . . .

• is historically important
• is easy to implement
• has interesting theoretical properties (convergence), but these are of very limited practical relevance
• achieves good performance often at the cost of substantial run-times
The Propositional Satisfiability Problem (SAT)

Simple SAT instance (in CNF):

\[(a \lor b) \land (\neg a \lor \neg b)\]

\(\sim\) satisfiable, two models:

\[a = \text{true}, b = \text{false}\]

\[a = \text{false}, b = \text{true}\]
Tabu Search

- Combinatorial search technique which heavily relies on the use of an explicit memory of the search process [Glover 1989, 1990] to guide search process
- memory typically contains only specific attributes of previously seen solutions
- simple tabu search strategies exploit only short term memory
- more complex tabu search strategies exploit long term memory
Tabu search – exploiting short term memory

- in each step, move to best neighboring solution although it may be worse than current one
- to avoid cycles, *tabu search* tries to avoid revisiting previously seen solutions by basing the memory on attributes of recently seen solutions
- tabu list stores attributes of the $tl$ most recently visited
- solutions; parameter $tl$ is called *tabu list length* or *tabu tenure*
- solutions which contain tabu attributes are forbidden
Tabu
Example: Tabu Search for SAT / MAX-SAT

- **Neighborhood**: assignments which differ in exactly one variable instantiation
- **Tabu attributes**: variables
- **Tabu criterion**: flipping a variable is forbidden for a given number of iterations
- **Aspiration criterion**: if flipping a tabu variable leads to a better solution, the variable’s tabu status is overridden

[Hansen & Jaumard 1990; Selman & Kautz 1994]
• Bart Selman, Cornell

www-verimag.imag.fr/~maler/TCC/selman-tcc.ppt

*Ideas from physics, statistics, combinatorics, algorithmics ...*

*The Boolean Satisfiability Problem: Theory and Practice*

*Bart Selman*
*Cornell University*

Joint work with Carla Gomes.
Fundamental challenge: Combinatorial Search Spaces

• *Significant progress in the last decade.*

• **How much?**

  • For propositional reasoning:
  • -- We went from 100 variables, 200 clauses (early 90’s)
  • to 1,000,000 vars. and 5,000,000 constraints in
  • 10 years. Search space: from $10^{30}$ to $10^{300,000}$.

  • -- Applications: Hardware and Software Verification,
  • Test pattern generation, Planning, Protocol Design,
  • Routers, Timetabling, E-Commerce (combinatorial
  • auctions), etc.
• How can deal with such large combinatorial spaces and
  still do a decent job?

• I’ll discuss recent formal insights into
  combinatorial search spaces and their
  practical implications that makes searching
  such ultra-large spaces possible.

• Brings together ideas from physics of disordered systems
  (spin glasses), combinatorics of random structures, and
  algorithms.

  • But first, what is BIG?
What is BIG?

Consider a real-world Boolean Satisfiability (SAT) problem

The instance bmc.ibm-6.cnf, IBM LSU 1997:

```
p cnf
-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

I.e., \((\neg x_1) \lor x_7\)
\((\neg x_1) \lor x_6\)

etc.

\(x_1, x_2, x_3, \text{etc. our Boolean variables (set to True or False)}\)

Set \(x_1\) to False ??
clauses / constraints are getting more interesting…

Note $x_1$ …
4000 pages later:

10236 -10050 0
10236 -10051 0
10236 -10235 0
10008 10009 10010 10011 10012 10013 10014
10015 10016 10017 10018 10019 10020 10021
10022 10023 10024 10025 10026 10027 10028
10029 10030 10031 10032 10033 10034 10035
10036 10037 10038 10039 10040 10041 10042 10043
10044 10045 10046 10047 10048 10049 10050 10051 10235 -10236 0
10237 -10008 0
10237 -10009 0
10237 -10010 0

...
Finally, 15,000 pages later:

\[
\begin{align*}
-7 & 260 0 \\
7 & -260 0 \\
1072 & 1070 0 \\
-15 & -14 -13 -12 -11 -10 0 \\
-15 & -14 -13 -12 -11 10 0 \\
-15 & -14 -13 -12 11 -10 0 \\
-15 & -14 -13 -12 11 10 0 \\
-7 & -6 -5 -4 -3 -2 0 \\
-7 & -6 -5 -4 -3 2 0 \\
-7 & -6 -5 -4 3 -2 0 \\
-7 & -6 -5 -4 3 2 0 \\
185 & 0 \\
\end{align*}
\]

Combinatorial search space of truth assignments: \[2^{50000} \approx 3.160699437 \cdot 10^{15051}\]

Current SAT solvers solve this instance in approx. 1 minute!
Progress SAT Solvers

<table>
<thead>
<tr>
<th>Instance</th>
<th>Posit' 94</th>
<th>Grasp' 96</th>
<th>Sato' 98</th>
<th>Chaff' 01</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssa2670-136</td>
<td>40,66s</td>
<td>1,2s</td>
<td>0,95s</td>
<td>0,02s</td>
</tr>
<tr>
<td>bf1355-638</td>
<td>1805,21s</td>
<td>0,11s</td>
<td>0,04s</td>
<td>0,01s</td>
</tr>
<tr>
<td>pret150_25</td>
<td>&gt;3000s</td>
<td>0,21s</td>
<td>0,09s</td>
<td>0,01s</td>
</tr>
<tr>
<td>dubois100</td>
<td>&gt;3000s</td>
<td>11,85s</td>
<td>0,08s</td>
<td>0,01s</td>
</tr>
<tr>
<td>aim200-2_0-no-1</td>
<td>&gt;3000s</td>
<td>0,01s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td>2dlx___bug005</td>
<td>&gt;3000s</td>
<td>&gt;3000s</td>
<td>&gt;3000s</td>
<td>2,9s</td>
</tr>
<tr>
<td>c6288</td>
<td>&gt;3000s</td>
<td>&gt;3000s</td>
<td>&gt;3000s</td>
<td>&gt;3000s</td>
</tr>
</tbody>
</table>

Source: Marques Silva 2002
• From academically interesting to practically relevant.

• We now have regular SAT solver competitions.
  • Germany ’89, Dimacs ’93, China ’96, SAT-02, SAT-03, SAT-04, SAT05.

• E.g. at SAT-2004 (Vancouver, May 04):
  • --- 35+ solvers submitted
  • --- 500+ industrial benchmarks
  • --- 50,000+ instances available on the WWW.
Figure 2. Performance Evolution of the Best SAT Solvers from 2002 to 2011.

The farther to the right the data points are, the better the solver.
Real-World Reasoning
Tackling inherent computational complexity

- High-Performance Reasoning
- Temporal/uncertainty reasoning
- Strategic reasoning/Multi-player

Example domains cast in propositional reasoning system (variables, rules).

- Car repair diagnosis
- Deep space mission control
- Chess
- Military Logistics

No. of atoms on earth $10^{47}$
Seconds until heat death of sun
Protein folding calculation (petaflop-year)

Worst Case complexity

Variables

Rules (Constraints)

Technology Targets

DARPA Research Program
Why all eggs in one basket?

• Why would we try to use only one point in search space?

• Try to use many and increase the search space “breadth”

• How to combine results?
Genetic Algorithms: A Tutorial

“Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime.”

- Salvatore Mangano
Computer Design, May 1995
The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970’s)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems
The Genetic Algorithm (cont.)

- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles
Components of a GA

A problem to solve, and ...

- Encoding technique \((\text{gene, chromosome})\)
- Initialization procedure \((\text{creation})\)
- Evaluation function \((\text{environment})\)
- Selection of parents \((\text{reproduction})\)
- Genetic operators \((\text{mutation, recombination})\)
- Parameter settings \((\text{practice and art})\)
Simple Genetic Algorithm

{ 
    initialize population;
    evaluate population;
    while TerminationCriteriaNotSatisfied
    {
        select parents for reproduction;
        perform recombination and mutation;
        evaluate population;
    }
}
The GA Cycle of Reproduction

- **reproduction**
  - parents
  - population
    - evaluated children
    - children
  - discard
    - deleted members
  - modification
    - modified children
  - evaluation
    - children

Wendy Williams
Metaheuristic Algorithms

Genetic Algorithms: A Tutorial
Genetic algorithms

• How to generate the next generation.
  • **1) Selection:** we select a number of states from the current generation. (we can use the fitness function in any reasonable way)
  • **2) crossover:** select 2 states and reproduce a child.
  • **3) mutation:** change some of the genues.
Example

= stochastic local beam search + generate successors from pairs of states
8-queen example

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components
Summary: Genetic Algorithms

Genetic Algorithms

• use populations, which leads to increased search space exploration
• allow for a large number of different implementation choices
• typically reach best performance when using operators that are based on problem characteristics
• achieve good performance on a wide range of problems
Example application: evolving checkers players (Fogel’02)

- Neural nets for evaluating future values of moves are evolved
- NNs have fixed structure with 5046 weights, these are evolved + one weight for “kings”
- Representation:
  - vector of 5046 real numbers for object variables (weights)
  - vector of 5046 real numbers for $\sigma$‘s
- Mutation:
  - Gaussian, lognormal scheme with $\sigma$-first
  - Plus special mechanism for the kings’ weight
- Population size 15
Example application: evolving checkers players (Fogel’02)

- Tournament size $q = 5$
- Programs (with NN inside) play against other programs, no human trainer or hard-wired intelligence
- After 840 generation (6 months!) best strategy was tested against humans via Internet
- Program earned “expert class” ranking outperforming 99.61% of all rated players
Mastering the game of Go without human knowledge

David Silver\textsuperscript{1*}, Julian Schrittwieser\textsuperscript{1*}, Karen Simonyan\textsuperscript{1*}, Ioannis Antonoglou\textsuperscript{1}, Aja Huang\textsuperscript{1}, Arthur Guez\textsuperscript{1}, Thomas Hubert\textsuperscript{1}, Lucas Baker\textsuperscript{1}, Matthew Lai\textsuperscript{1}, Adrian Bolton\textsuperscript{1}, Yutian Chen\textsuperscript{1}, Timothy Lillicrap\textsuperscript{1}, Fan Hui\textsuperscript{1}, Laurent Sifre\textsuperscript{1}, George van den Driessche\textsuperscript{1}, Thore Graepel\textsuperscript{1} & Demis Hassabis\textsuperscript{1}

A long-standing goal of artificial intelligence is an algorithm that learns, \textit{tabula rasa}, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo’s own move selections and also the winner of AlphaGo’s games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting \textit{tabula rasa}, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.
The GA Cycle of Reproduction

- **reproduction**
  - parents
  - population
  - delete members
- **modification**
  - children
  - evaluated children
  - modified children
  - evaluation
- **discard**
  - evaluated children
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 ...
Reproduction

Parents are selected at random with selection chances biased in relation to chromosome evaluations.
Chromosome Modification

- Modifications are stochastically triggered
- Operator types are:
  - Mutation
  - Crossover (recombination)
**Mutation: Local Modification**

Before: \((1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0)\)

After: \((0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0)\)

Before: \((1.38 \ -69.4 \ 326.44 \ 0.1)\)

After: \((1.38 \ -67.5 \ 326.44 \ 0.1)\)

- Causes movement in the search space (local or global)
- Restores lost information to the population
Crossover: Recombination

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

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)
With a probability of 0.5, children have 50% genes from first parent and 50% of genes from second parent even with randomly chosen crossover points.
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
An Abstract Example

Distribution of Individuals in Generation 0

Distribution of Individuals in Generation N
“The Gene is by far the most sophisticated program around.”

- Bill Gates, Business Week, June 27, 1994
A 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
Representation

Representation is an ordered list of city numbers known as an order-based GA.

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

Crossover combines inversion and recombination:

\[
\begin{array}{c}
\text{Parent1} (3 & 5 & 7 & 2 & 1 & 6 & 4 & 8) \\
\text{Parent2} (2 & 5 & 7 & 6 & 8 & 1 & 3 & 4) \\
\hline
\text{Child} (5 & 8 & 7 & 2 & 1 & 6 & 3 & 4)
\end{array}
\]

This operator is called the Order1 crossover.
Mutation

Mutation involves reordering of the list:

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

After: (5 8 6 2 1 7 3 4)
TSP Example: 30 Cities
Solution \( i \) (Distance = 941)
Solution \( j(\text{Distance} = 800) \)
Solution \textsubscript{k}(Distance = 652)
Best Solution (Distance = 420)
Overview of Performance

TSP30 - Overview of Performance

Generations (1000)

Best
Worst
Average
Considering the GA Technology

“Almost eight years ago ... people at Microsoft wrote a program [that] uses some genetic things for finding short code sequences. Windows 2.0 and 3.2, NT, and almost all Microsoft applications products have shipped with pieces of code created by that system.”

- Nathan Myhrvold, Microsoft Advanced Technology Group, Wired, September 1995
Issues for GA Practitioners

● Choosing basic implementation issues:
  ♦ representation
  ♦ population size, mutation rate, ...
  ♦ selection, deletion policies
  ♦ crossover, mutation operators

● Termination Criteria

● Performance, scalability

● Solution is only as good as the evaluation function (often hardest part)
Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed
Benefits of Genetic Algorithms (cont.)

- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use
When to Use a GA

- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements
## Some GA Application Types

<table>
<thead>
<tr>
<th>Domain</th>
<th>Application Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>gas pipeline, pole balancing, missile evasion, pursuit</td>
</tr>
<tr>
<td>Design</td>
<td>semiconductor layout, aircraft design, keyboard configuration, communication networks</td>
</tr>
<tr>
<td>Scheduling</td>
<td>manufacturing, facility scheduling, resource allocation</td>
</tr>
<tr>
<td>Robotics</td>
<td>trajectory planning</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>designing neural networks, improving classification algorithms, classifier systems</td>
</tr>
<tr>
<td>Signal Processing</td>
<td>filter design</td>
</tr>
<tr>
<td>Game Playing</td>
<td>poker, checkers, prisoner's dilemma</td>
</tr>
<tr>
<td>Combinatorial Optimization</td>
<td>set covering, travelling salesman, routing, bin packing, graph colouring and partitioning</td>
</tr>
</tbody>
</table>
Review
4 main types of Evolutionary Algorithms

- Genetic Algorithm - John Holland
- Genetic Programming - John Koza
- Evolutionary Programming - Lawerence Fogel
- Evolutionary Strategies - Ingo Rechenberg
Genetic Algorithms

• Most widely used
• Robust
• uses 2 separate spaces
  – search space - coded solution (genotype)
  – solution space - actual solutions (phenotypes)
• Genotypes must be mapped to phenotypes before the quality or fitness of each solution can be evaluated
Genetic Programming

- Specialized form of GA
- Manipulates a very specific type of solution using modified genetic operators
- Original application was to design computer program
- Now applied in alternative areas eg. Analog Circuits
- Does not make distinction between search and solution space.
- Solution represented in very specific hierarchical manner.
Genetic Programming

- Evolves more complex structures - programs, Lisp code, neural networks
- Start with random programs of functions and terminals (data structures)
- Execute programs and give each a fitness measure
- Use crossover to create new programs, no mutation
- Keep best programs
- For example, place lisp code in a tree structure, functions at internal nodes, terminals at leaves, and do crossover at sub-trees - always legal in Lisp
Evolutionary Strategies

• Like GP no distinction between search and solution space
• Individuals are represented as real-valued vectors.
• Simple ES
  – one parent and one child
  – Child solution generated by randomly mutating the problem parameters of the parent.
• Susceptible to stagnation at local optima
Evolutionary Strategies (cont’d)

- Slow to converge to optimal solution
- More advanced ES
  - have pools of parents and children
- Unlike GA and GP, ES
  - Separates parent individuals from child individuals
  - Selects its parent solutions deterministically
General Idea of Evolutionary Algorithms

Figure 1.17 The general architecture of evolutionary algorithms (GAEA).
# Summary

<table>
<thead>
<tr>
<th></th>
<th>ES</th>
<th>EP</th>
<th>GA</th>
<th>GP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representation</strong></td>
<td>Real-valued</td>
<td>Real-valued</td>
<td>Binary-Valued</td>
<td>Lisp S-expressions</td>
</tr>
<tr>
<td><strong>Self-Adaptation</strong></td>
<td>Standard deviations and covariances</td>
<td>Variance</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td><strong>Fitness</strong></td>
<td>Objective function values</td>
<td>Scaled objective function value</td>
<td>Scaled objective function value</td>
<td>Scaled objective function value</td>
</tr>
<tr>
<td><strong>Mutation</strong></td>
<td>Main operator</td>
<td>Only operator</td>
<td>Background operator</td>
<td>Background operator</td>
</tr>
<tr>
<td><strong>Recombination</strong></td>
<td>Different variants, important for self-adaptation</td>
<td>None</td>
<td>Main Operator</td>
<td>Main Operator</td>
</tr>
<tr>
<td><strong>Selection</strong></td>
<td>Deterministic extintive</td>
<td>Probabilistic, extintive</td>
<td>Probabilistic, preservative</td>
<td>Probabilistic, preservative</td>
</tr>
</tbody>
</table>
Evolutionary design

Sims - Evolved Virtual Creatures, Evolution Simulation

[Video](https://www.youtube.com/watch?v=JBgG_VSP7f8) - Dec 6, 2008 - Uploaded by MediaArtTube
Karl Sims - Evolved Virtual Creatures, Evolution Simulation 1994 Something like this on a small self...

Evolved Virtual Creatures - Internet Archive

[Archive](https://archive.org) - SIGGRAPH
Karl Sims Evolved Virtual Creatures (1994) ... A population of several hundred creatures is created within a...
Evolutionary design

• Karl Sims Evolved Virtual Creatures (1994)
  – http://www.youtube.com/watch?v=F0OHycypSG8
  – http://video.google.com/videoplay?docid=7219479512410540649#
  – course work - 2005


• http://vimeo.com/7074089
Google's DeepMind AI just taught itself to walk

4,629,765 views
40 days

AlphaGo Zero surpasses all other versions of AlphaGo and, arguably, becomes the best Go player in the world. It does this entirely from self-play, with no human intervention and using no historical data.
TPU – Tensor processing unit:
https://en.wikipedia.org/wiki/Tensor_processing_unit
Could you paint a replica of the Mona Lisa using only 50 semi transparent polygons?
Mona Lisa

- [Link](https://www.google.ee/search?q=mona+lisa+evolution&newwindow=1&tbm=vid&source=lncms)
Figure 9.1 *Mutator* keeps a bank of genes and their forms (generated by *Form Grow*), which it displays to the artist. Based on judgements made by the artist, *Mutator* generates and displays new forms, assisting the artist to search for interesting forms and bank the results.
structure expression:
  horn
    ribs (gene1)
    grow (gene2)
    stack (gene3)
    bend (gene4)
    twist (gene5)

corresponding gene vector:
< gene1, gene2, gene3, gene4, gene5 >

**Figure 9.5** An example of a structure expression (created by the artist) and its corresponding gene vector (to be evolved by *Mutator*).
Figure 9.6 A frame of nine mutations. The parent is in the centre surrounded by offspring.
A. Very low mutation rate
B. Low mutation rate
C. Medium mutation rate
D. High mutation rate
inbreeding

distant marriage
Figure 9.16 Extract from an evolutionary tree. The tree has become too large to display clearly, so the artist has restricted the display to include only frames between one level above and one level below the current frame. Cousin frames are not displayed.
Figure 9.24 The forms layed out in a continuous *Mutator* session much as they would be in an animation such as the film ‘*Mutations*’.
Conclusions

Question: ‘If GAs are so smart, why ain’t they rich?’

Answer: ‘Genetic algorithms are rich - rich in application across a large and growing number of disciplines.’

- David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning
• Games: Spore (2007)

http://www.ted.com/talks/will_wright_makes_toys_that_make_worlds.html

• http://www.gametrailers.com/user-movie/spore-14min-2007-demonstration/86368

• http://eu.spore.com/home.cfm?lang=en
Components of a GA

A problem to solve, and ...

• **Encoding technique**  \((\text{gene, chromosome})\)
• Initialization procedure \((\text{creation})\)
• **Evaluation function**  \((\text{environment})\)
• Selection of parents \((\text{reproduction})\)
• Genetic operators \((\text{mutation, recombination})\)
• **Parameter settings**  \((\text{practice and art})\)
Ant Colony Optimization (ACO)

Strong or weak pheromone
TSP
More on ACO

• Foraging – looking for new alternative routes

• It can work on dynamic systems, adopting continuously to changes in the environment
Differential Evolution

• Real-valued chromosomes

• $X = (x_1, x_2, x_3, ..., x_n)$

Potential neighbourhood, mutations…
Evolutionary Approaches

Differential Evolution

- repeated updating of solutions
- changes depend on relative positions
- magnitude of changes depends on “diversity” within the population
Evolutionary Approaches

Differential Evolution

- repeated updating of solutions
- changes depend on relative positions
- magnitude of changes depends on “diversity” within the population
Evolutionary Approaches

Implementing Differential Evolution

1. randomly initialize population, $x_p$, $p = 1 \ldots P$

2. REPEAT

   a) for each member $p$, create 1 offspring $o$

   $$x_o[i] := \begin{cases} x_m[i] + F \cdot (x_{m_2}[i] - x_{m_3}[i]) & \text{with prob. } \pi \\ x_p[i] & \text{with prob. } 1 - \pi \end{cases}$$

   b) decide over replacement ("tournament"): if $f(x_o) < f(x_p)$ then

   $$x_p := x_o$$

   UNTIL halting criterion met
Evolutionary Approaches

Implementing Differential Evolution (cont’d)

- continuous search space

- calibrating
  - population size
  - scaling parameter
  - cross-over probability, $\pi$

- extensions
  - jitter (*add noise to difference vector and / or F*)
  - include “elitist” (*best solution so far*)
  - mapping functions for constraints or search space
Robust Regression
(Differential Evolution)

Fitting a regression line using minimum median error as a measure.

\[ aX + bY + c = 0 \]
\[ Y = aX + c \]

Find \( a \) and \( c \)
Mean squared error (MSE)

Median of square error?

Robust Regression

least quantile of squares
(Gilli, Maringer and Schumann, 2011)

$$\min_{\beta} e_{(\alpha N)}^2 \quad \text{where} \quad e = X\beta - y$$

$$e_{(j-1)}^2 \leq e_{(j)}^2, \quad j = 2..N$$
Differential Evolution: project

Fit **any polynomial**, use mean or median, add **MDL based identification of the degree of polynomial**

\[ A_n X^n + A_{n-1} X^{n-1} + \ldots + A_1 X + A_0 \]
Another Population Based Approach

“Particle Swarm Optimization”
(J. Kennedy and R. Eberhart (1995))

- particles move through solution space
- components of direction (“velocity” $v$)
  - current direction (“inertia”)
  - are drawn to “good” solutions
    (personal best ($x_p$) & overall best ($x_g$) so far)

$$v := v + c_1 \cdot \tilde{z}_1 (x_p - x) + c_2 \cdot \tilde{z}_2 (x_g - x)$$

$$x := x + v, \quad \tilde{z}_i \sim U(0, 1)$$
Another Population Based Approach

Implementing PSO

1. randomly initialize population, $x_p$ and $v_p$
2. REPEAT
   a) for each member $p$, create 1 offspring $o$
      
      $v := v + c_1 \cdot \tilde{z}_1 (x_p - x) + c_2 \cdot \tilde{z}_2 (x_g - x)$
      
      $x := x + v, \quad \tilde{z}_i \sim U(0, 1)$
   b) check for ...
      
      – new personal best
      – new global best

UNTIL halting criterion met
Another Population Based Approach

Implementing PSO (cont’d)

- continuous search space

- calibrating
  - population size
  - weights $c_1$, $c_2$

- extensions
  - mapping functions for constraints or search space
  - decay on speed
  - additional relative positions
Heuristic Optimization Methods

Summary & Conclusions

— deterministic + non-deterministic elements
  • generation of new candidate solutions
  • acceptance of new candidate solutions
— general purpose

— selection and implementation issues
  • calibration
  • constraint satisfaction
  • hybrid methods
Summary

• **Encoding** – search space and search steps
• **Evaluation**
• **Optimisation goal**

• Heuristics – various ideas – what are the traces of “good partial solutions”