Advanced Algorithmics (6EAP)
Search and meta-heuristics
Jaak Vilo
2019 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):

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

\[\neg\] satisfiable, two models:

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

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

TSP, NN suboptimal

Cost of Nearest Neighbor Path, ABDECA = 550

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>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect information</td>
<td>backgammon monopoly</td>
</tr>
<tr>
<td>chess, checkers, go, othello</td>
<td>bridge, poker, scrabble nuclear war</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Plot</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Himmelblau function</td>
<td><img src="image" alt="Himmelblau Function" /></td>
<td>( f(x) = A \sum_{i=1}^{n} (x_i^2 - 4x_i + 4x_i + 2) ) where ( A = 3 )</td>
</tr>
<tr>
<td>Aizerman's function</td>
<td><img src="image" alt="Aizerman's Function" /></td>
<td>( f(x) = 2 \cos(\sqrt{x_1^2 + x_2^2}) - \sin(3(x_1^2 + x_2^2)) ) for ( x_1, x_2 \in [0, 10] )</td>
</tr>
<tr>
<td>Sphere function</td>
<td><img src="image" alt="Sphere Function" /></td>
<td>( f(x) = \sum_{i=1}^{n} x_i^2 )</td>
</tr>
<tr>
<td>Rosenbrock function</td>
<td><img src="image" alt="Rosenbrock Function" /></td>
<td>( f(x) = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] ) for ( x_1, x_2 \in [0, 10] )</td>
</tr>
</tbody>
</table>

An interesting constrained numerical optimization test case emerged recently: the problem (Himmelblau, 1994) is to maximize a function:

\[
G_2(x) = \left( \sum_{i=1}^{n} x_i \right)^2 - \prod_{i=1}^{n} \cos^2(x_i) / \sqrt{\sum_{i=1}^{n} x_i^2},
\]

where \( 0 \leq x_i \leq 10 \) and \( \prod x_i \geq 0.75 \).
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, \((x_0, 0)\) 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**

- Local search algorithms
  - Hill-climbing
  - Simulated annealing
  - Particle Swarm
  - Ant colony optimisation
- Genetic algorithms
  - Genetic algorithms
    - Evolutionary strategies
    - Evolutionary algorithms
    - Genetic algorithms
    - Simulated annealing
    - Dynamic programming
- Parallel and distributed algorithms
  - Parallel and distributed algorithms
    - Parallel algorithms
    - Distributed algorithms
- Heuristic algorithms
  - Heuristic algorithms
    - Heuristic algorithms
    - Heuristic algorithms
    - Heuristic algorithms
Greedy

- Always choose the seemingly best step

**Greedy Set Covering Algorithm**

**Greedy 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**

**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 \(X\): \(X = \bigcup_{S \in C} S\)

\( \mathcal{F} \) is \(\Pi\)-Complete

source: 91.503 textbook Cormen et al.

**Greedy**

select set that covers the most uncovered elements

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

**Proof**

- 1. Create \(H_d\) for each set \(S_i\) selected
- 2. Assume \(x\) is covered for the first time by \(S_i\)
- 3. \(c_x = \frac{1}{|\bigcup_{S_j \supseteq S_i}(S_j \cup \cdots \cup S_i)|} \)

Number of elements covered for first time by \(S_i\)

---

**Harmonic number**

From Wikipedia, the free encyclopedia

The harmonic numbers have 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 \(n\) positive integers:

\[ H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \]

This also equals the inverse of the harmonic mean of \(n\) positive numbers.

Harmonic numbers were studied in antiquity and are important in various branches of modern mathematics.

When the value of a large quantity of items is ranked in a Descr or distribution, the total value of the most valuable items is the \(n\)th harmonic number. This leads to a study of a shaping correlation in the Laws of the Theory of Network Value.
Proof of approximation

• [http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/wan-ba-notes.pdf](http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/wan-ba-notes.pdf)

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) \( (SLD – \text{shortest linear distance}) \)

- **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 \( G \).

  - \( f(G_2) = g(G_2) \) 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=19h1g2hbh8

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 ↔ local search
  - deterministic ↔ stochastic
  - sequential ↔ 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: $O(n^2)$
- worst case performance: $NN(x)/OPT(x) \leq 0.5([\log_2 n] + 1)$
- other construction heuristics for TSP are available
Nearest neighbor tour through 532 US cities

TSP instance: shortest round trip through 532 US cities

https://www.cs.duke.edu/courses/fall08/cps230/Lectures/L25.pdf

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 $c : E \rightarrow \mathbb{Z}_+$. A tour in this graph is a Hamiltonian cycle and the problem is finding the tour, $A$, with minimum total cost, $c(A) = \sum_{(u,v) \in A} c(uv)$. Let us first assume that the cost function satisfies the triangle inequality, $c(uv) \leq c(uw) + c(wv)$ 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 $T$.

My current best is 27486.1994046355 (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

http://www.improbable.com/2016/10/19/a-shortest-possible-walking-tour-through-the-pubs-of-the-uk/

A shortest-possible walking tour through the pubs of the UK

http://www.math.uwaterloo.ca/tsp/

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

Hill climbing function

\[
\text{function } \text{Hill-Climbing}(\text{problem}) \text{ returns a solution state}
\]

\[
current \leftarrow \text{Make-Node}(\text{Initial-State}(\text{problem}))
\]

\[
\text{loop do}
\]

\[
\text{next} \leftarrow \text{a highest-valued successor of current}
\]

\[
\text{if Value}[\text{next}] < \text{Value}[\text{current}] \text{ then return current}
\]

\[
current \leftarrow \text{next}
\]

\[
\text{end}
\]

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**

- 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)
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 the hill-climbing methods — 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]

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, $T_{n+1} = \alpha \cdot T_n$, $\alpha = 0.1, \ldots$)
- 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


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 ← V-Value[next] - V-Value[current]

if ΔE > 0 then current ← next

else current ← next with probability e^ΔE/T

end

Example application to the TSP [Johnson & McGeoch 1997]

baseline implementation:
- start with random initial solution
- use 2-exchange neighborhood
- simple annealing schedule;

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.
Summary-Simulated Annealing

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):
\[(a \lor b) \land (\neg a \lor \neg b)\]
\[\sim\text{satisfiable, two models:}\]
\[a = \text{true}, b = \text{false}\]
\[a = \text{false}, b = \text{true}\]

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
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]

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 k=10: k=10, out, IBM LDB 1997:

```
0 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0

0 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 0 1 1 1 0 0
```

i.e. (not x_1) or x_7

((not x_1) or x_6)

etc.

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

Set x_1 to False ??
10 pages later:

\[(x_{177} \text{ or } x_{160} \text{ or } x_{161} \text{ or } x_{153} \ldots \text{ or } x_{33} \text{ or } x_{25} \text{ or } x_{17} \text{ or } x_{9} \text{ or } x_{1} \text{ or } \neg x_{185})\]

clauses / constraints are getting more interesting...

Note \(x_1\) ...

4000 pages later:

Finally, 15,000 pages later:

Combinatorial search space of truth assignments:

\[2^{100} = 3.6088 	imes 10^{30}\]

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-no1</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.
**Real-World Reasoning**

Tackling inherent computational complexity

---

**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?

---

**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

---

**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*
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)

Simple Genetic Algorithm
{
  initialize population;
  evaluate population;
  while TerminationCriteriaNotSatisfied
  {
    select parents for reproduction;
    perform recombination and mutation;
    evaluate population;
  }
}

The GA Cycle of Reproduction

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 genes.

Example
- 8-queen example
  - stochastic local beam search - generate successors from pairs of states
  - GAs require states encoded as strings (GPs use programs)
  - Crossover helps if 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 n's
- Mutation:
  - Gaussian, lognormal scheme with n-first
  - Plus special mechanism for the kings' weight
- Population size 15

The GA Cycle of Reproduction

Population
- Chromosomes could be:
  - Bit strings
  - Real numbers
  - Permutations of element
  - Lists of rules
  - Program elements
  - Any data structure...

2017: GO

Mastering the game of Go without human knowledge
David Silver*, Julian Schrittwieser*, Ioannis Antonoglou*, Aja Huang, Arthur Guez*, Thomas Hubert-Lang, David Lille, Max Jaderberg, Tiacayn Chen, Volodya Gues, Aurélien Cordonnier, David Sifre and Demis Hassabis

A long-standing goal of artificial intelligence is an algorithm that learns, makes new, superhuman proficiency in all challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. This team used a hybrid approach that included both machine learning and human expertise. This hybrid approach involves the use of deep neural networks trained by games played by human Go experts, and by reinforcement learning from self-play. In this article we introduce the key ideas of our approach and describe the system architecture of AlphaGo in more detail. We also describe how AlphaGo’s machine learning modules were designed to support the knowledge acquired by its human experts. This research builds on the success of the AlphaGo system, enabling higher quality moves and more effective play as it continues to improve. Starting today, our new program AlphaGo Zero achieved superhuman performance, winning 60-0 against the previously published, champion-defeating AlphaGo.
### 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 1 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

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)
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

A Simple Example

"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:

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
Mutation involves reordering of the list:

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

TSP Example: 30 Cities
Solution i (Distance = 941)

Solution j (Distance = 800)

Solution k (Distance = 652)
Best Solution (Distance = 420)

Overview of Performance

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</td>
</tr>
<tr>
<td>Scheduling</td>
<td>configuration, communication networks</td>
</tr>
<tr>
<td>Robotics</td>
<td>manufacturing, facility scheduling, resource allocation</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>designing neural networks, improving classification</td>
</tr>
<tr>
<td>Signal Processing</td>
<td>algorithms, classifier systems</td>
</tr>
<tr>
<td>Game Playing</td>
<td>fiber design</td>
</tr>
<tr>
<td>Combinatorial Optimization</td>
<td>set covering, travelling salesman, routing, bin packing, graph colouring and partitioning</td>
</tr>
</tbody>
</table>

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.

Review

4 main types of Evolutionary Algorithms

- Genetic Algorithm - John Holland
- Genetic Programming - John Koza
- Evolutionary Programming - Lawrence Fogel
- Evolutionary Strategies - Ingo Rechenberg
**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

![Image](image1.png)

- **Representation**: ES = real-valued, EP = binary-valued, OA = string, GP = tree structure
- **Selection**: ES = deterministic, EP = probabilistic, OA = probabilistic, GP = probabilistic
- **Crossover**: ES = always possible, EP = generally not, OA = generally not
- **Mutation**: ES = simple, EP = generally not, OA = generally not
- **Adaptation**: ES = generally not, EP = generally not, OA = generally not

![Image](image2.png)

### Summary

<table>
<thead>
<tr>
<th>Representation</th>
<th>ES</th>
<th>EP</th>
<th>OA</th>
<th>GP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>real-valued</td>
<td>real-valued</td>
<td>binary-valued</td>
<td>string</td>
</tr>
<tr>
<td>Self-Adaptation</td>
<td>standard deviations and correlations</td>
<td>variance</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Fitness</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>Mutation</td>
<td>main operator</td>
<td>only operator</td>
<td>background operator</td>
<td>main operator</td>
</tr>
<tr>
<td>Recombination</td>
<td>different variants, important for evolution</td>
<td>none</td>
<td>main operator</td>
<td>main operator</td>
</tr>
<tr>
<td>Selection</td>
<td>deterministic</td>
<td>probabilistic</td>
<td>probabilistic</td>
<td>probabilistic</td>
</tr>
</tbody>
</table>
Evolutionary design

- Karl Sims Evolved Virtual Creatures (1994)
  - http://www.youtube.com/watch?v=FOOHg3s5G8

- http://video.google.com/videoplay?docid=7219479512410540649#
- course work - 2005
- http://vimeo.com/7074089

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.

https://www.youtube.com/watch?v=gn4nRCC9TwQ

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

- [https://www.google.ee/search?q=mona+lisa+evolution&newwindow=1&tbm=vid&source=lms](https://www.google.ee/search?q=mona+lisa+evolution&newwindow=1&tbm=vid&source=lms)
Figure 3.16: Extract from an evolutionary tree. The tree has become too large to display clearly, so the artist has restricted the display to include only branches between one level above and one level below the current frame. cousin frames are not displayed.
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 (gene, chromosome)
• Initialization procedure (creation)
• Evaluation function (environment)
• Selection of parents (reproduction)
• Genetic operators (mutation, recombination)
• Parameter settings (practice and art)

Ant Colony Optimization (ACO)

1991, M. Dorigo proposed the Ant System in his doctoral thesis

Strong or weak pheromone

More on ACO

• Foraging – looking for new alternative routes

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

TSP
**Differential Evolution**

- Real-valued chromosomes

\[ X = (x_1, x_2, x_3, \ldots, x_n) \]

Potential neighborhood, mutations...

**Evolutionary Approaches**

**Differential Evolution**
- repeated updating of solutions
- changes depend on relative positions
- magnitude of changes depends on “diversity” within the population

**Implementing Differential Evolution**
1. randomly initialize population, \( x_p, p = 1 \ldots P \)
2. REPEAT
   a) for each member \( p \), create 1 offspring \( o \)
      \[ x_o[j] = \frac{1}{2} (x_{p1}[j] + x_{p2}[j] - x_{p3}[j]) \] with prob. \( p \)
      \[ x_o[j] = x_{p3}[j] \] with prob. \( 1-p \)
   b) decide over replacement (“tournament”):
      if \( f(x_o) < f(x_p) \) then
      \[ x_p := x_o \]
   UNTIL halting criterion met

**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)

Mean of square error
Median of square error?


Robust Regression

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

\[
\text{min}_{\beta} \epsilon_{ij} \text{ where } e = X\beta - y
\]

\[
\epsilon_{ij}^2 < \epsilon_{ij}^2, j = 2, N
\]

Differential Evolution: project

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

\[ A_n X^n + A_{n-1} X^{n-1} + \ldots + A_1 X + A_0 \]
**Summary**

- **Encoding** – search space and search steps
- **Evaluation**
- **Optimisation goal**
- **Heuristics** – various ideas – what are the traces of “good partial solutions”
Intelligent Machines

**Evolutionary algorithm outperforms deep-learning machines at video games**

Neural networks have garnered all the headlines, but a much more powerful approach to solving the enigma.

by Emerging Technology from the Acta - July 19, 2018

With all the excitement over neural networks and deep-learning techniques, it’s easy to imagine that the world of computer science consists of nothing but networks. Neural networks, after all, have begun to outperform humans in tasks such as object and face recognition and in games such as chess, Go, and various simple video games.