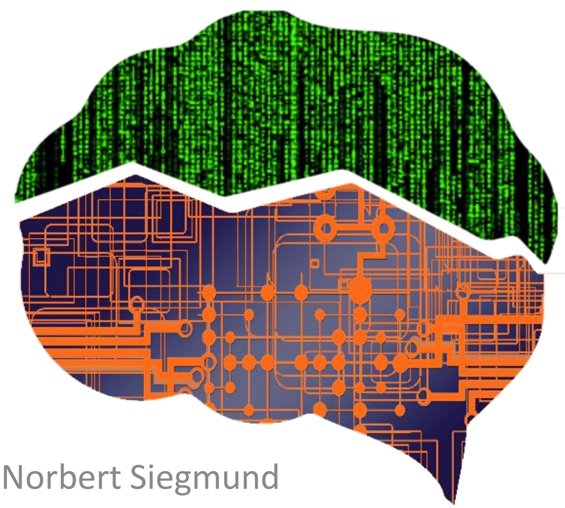
Machine Learning for Software Engineering

Single-State Meta-Heuristics



Prof. Dr.-Ing. Norbert Siegmund
Intelligent Software Systems

Bauhaus-Universität Weimar

Recap: Goal is to Find the Optimum

- Challenges of general optimization problems (not combinatorial for the moment):
  - Search space is too big
  - Too many solutions to compute
  - Even good heuristics for a *systematic* search are too costly in terms of performance and memory consumption
  - Note that we consider combinatorial optimization problems in later lectures based on the optimization approaches we learn next

- But, how to do optimization in a good-case scenario?

Gradient-based Optimization

- Given a cost function f(x), we can find the optimum via gradient ascent as long as we can compute the first derivative f'(x)
- Idea: Compute the slope at any given x and move up
$$x \leftarrow x + \alpha f'(x)$$
- With: α is a very small positive number controlling the extent of the change

- Generalization with $\vec{x}$ as the input vector:
$$\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$$

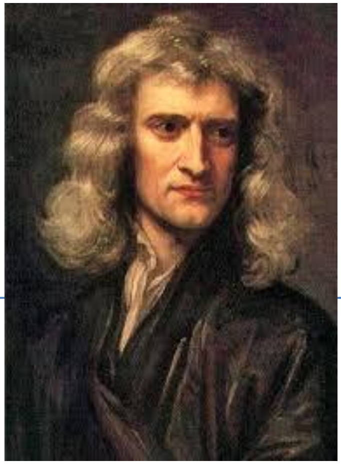
The gradient ∇ is a vector containing the derivative of each element of that dimension

Algorithm and Problems

$\vec{x} \leftarrow$ random initial vector
repeat
  $\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$
until $\vec{x}$ *is optimum or out of time*
return $\vec{x}$

- When do we know $\vec{x}$ is the optimum?
  - Slope is 0
  - Be ware of saddle points and minima!
- What is the convergence time?
  - Tuning α for convergence and against overshooting
- What else can we do?
  - Newton's Method: Directly compute extreme points with f''(x)

Newton's Method I

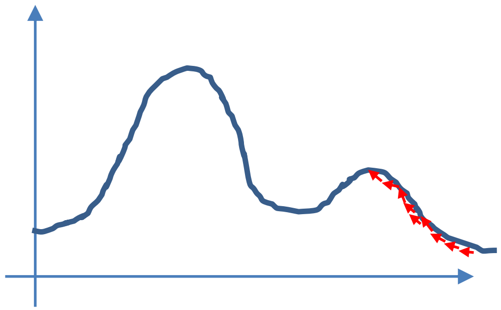


- One-dimensional case: $\vec{x} \leftarrow \vec{x} - \alpha \dfrac{f'(\vec{x})}{f''(\vec{x})}$
  - Dampens α as we get closer to zero slope
  - But, heads to any kind of zero slope (minima, maxima, saddle)
- Multi-dimensional version of the f''(x) is more complex:
  - Called: **Hessian** $H_f(\vec{x}) = \begin{bmatrix} \frac{\delta}{\delta x_1}\frac{\delta f}{\delta x_1} & \cdots & \frac{\delta}{\delta x_1}\frac{\delta f}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta}{\delta x_n}\frac{\delta f}{\delta x_1} & \cdots & \frac{\delta}{\delta x_n}\frac{\delta f}{\delta x_n} \end{bmatrix}$
  - Partial second derivative along each dimension

Newton's Method II

$\vec{x} \leftarrow$ random initial vector
repeat
$\quad \vec{x} \leftarrow \vec{x} - \alpha \left[H_f(\vec{x})\right]^{-1} \nabla f(\vec{x})$
until $\vec{x}$ *is optimum or out of time*
return $\vec{x}$

- Converges faster than regular gradient ascent
- Problems:
  - Caught in local optima, but goal is global optima



Local optimization algorithm!

Toward Global Optimization

- Two options: increase α or repeat gradient ascent in a loop and always start from a different random position

$\vec{x} \leftarrow$ random initial vector
$\vec{x}^* \leftarrow \vec{x}$
repeat
  repeat
    $\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$
  until $|| \nabla f(\vec{x}) || = 0$
  if $f(\vec{x}) > f(\vec{x}^*)$ **then**
    $\vec{x}^* \leftarrow \vec{x}$
  $\vec{x} \leftarrow$ random vector
until *out of time*
return $\vec{x}^*$

Finds local optimum

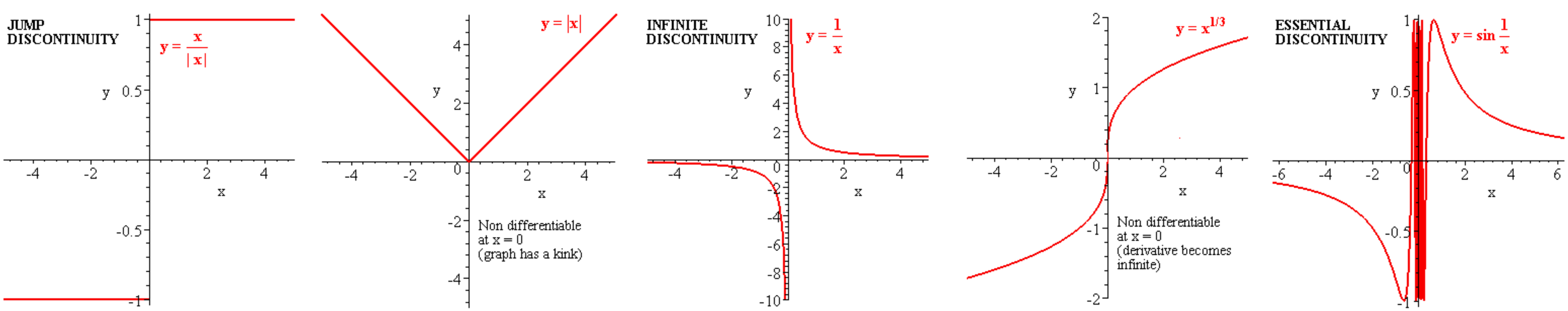
Finds the best local optimum, which is hopefully the global optimum

- Problem: $||\nabla f(\vec{x})|| = 0$ might never be exactly 0, so use a threshold: $-\epsilon < ||\nabla f(\vec{x})|| < \epsilon$

Shortcomings of Gradient Ascent

- Assumptions:
  - Ability to compute the first derivative
  - Often, we even don’t know the function (e.g., in black-box scenarios)!
  - We only know how to create, modify, and test a solution

- Does not work for non-differentiable functions



Solution: Thoughtfull Random Probing

- Idea: Randomly select a starting point in the search space and search based on a *given strategy* for the optimal solution
- The given strategy represents the *meta-heuristic*

- This lecture:
  - Know pros and cons of gradient-based optimization
  - Learn about *single-state* meta-heuristics
    - Local search
    - Global search
    - Hill climbing, simulated annealing, etc.

Heuristics

- Heuristic (greek: to find)
  – “involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods” Merriam-Webster dicitionary
- Why heuristics?
  – NP-hard problems including decision variables with many interdependencies
  – Nonlinear cost functions and constraints, even no mathematical functions (e.g., a cost function might be the execution of a program or asking an expert)
  – So, near-optimal solution might be just good enough

Meta-Heuristic

- Algorithms employing some degree of randomness to find "optimal" solutions to hard problems

- Applied to: "I know it when I see it" problems
  - In case when:
    - You don't know beforehand how the optimal solution looks like
    - You don't know how to find the optimal solution
    - The search space is too large and there is no domain heuristic
    - You can quantify the quality of a solution when you see it
- Two extremes:

Random search — Hill climbing

Assumptions of Meta-Heuristic Optimization

- We need to be able to do four steps:
  - **Initialization procedure**: Provide one or more initial candidate solutions
  - **Assessment procedure**: Assess the *quality* of a candidate solution
  - Make a *copy* of a candidate solution
  - **Modification procedure**: *Tweak* a candidate solution to produce a randomly slightly different candidate solution
- A **selection procedure** decides, which candidate solution to retain

Hill Climbing (Local Search)

- Idea:
  - Use only your local solution and evaluate your *neighbors* to find a better one
  - Repeat this step until no better neighbor exists
  - Similar to gradient ascent, but does not compute gradient

- Pros:
  - Requires few resources (current state and neighbors)
  - Finds local optimum (global is possible)
  - Useful if the search space is huge (even unlimited)

Hill-Climbing Algorithm

$S \leftarrow$ random initial $solution$ ← Initialization procedure
repeat
  $R \leftarrow Tweak(Copy(S))$ ← Modification procedure
  if $Quality(R) > Quality(S)$ **then** ← Assessment and selection procedure
    $S \leftarrow R$
until S is optimum or out of time
return S

- Observations:
  - Hill climbing is more general than gradient ascent
  - *Tweak* operation must rely on a stochastic/random process to find better candidate solutions
  - Strongly depends on "good" initialization

Variant: Steepest Ascent Hill Climbing

- Idea: Be more aggressive and parallelize by creating n tweaks to a candidate solution (like sampling the gradient)

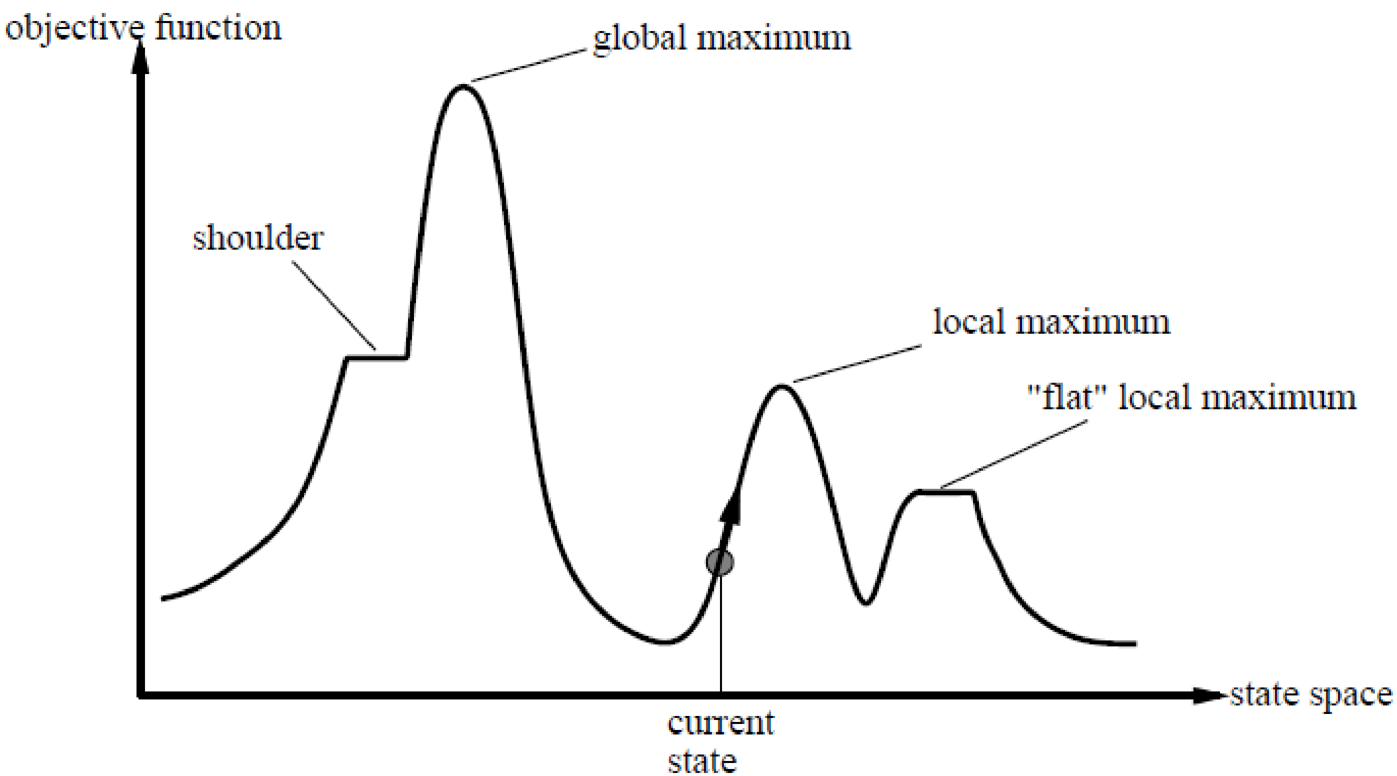
$n \leftarrow$ number of tweaks
$S \leftarrow$ random initial $solution$
repeat
  $R \leftarrow Tweak(Copy(S))$
  for n-1 times **do**
    $W \leftarrow Tweak(Copy(S))$
    if $(Quality(W) > Quality(R)$ **then**
      $R \leftarrow W$
  if $(Quality(R) > Quality(S)$ **then**
    $S \leftarrow R$
until S is optimum or out of time
return S

With replacement:

$n \leftarrow$ number of tweaks
$S \leftarrow$ random initial $solution$
$Best \leftarrow S$
repeat
  $R \leftarrow Tweak(Copy(S))$
  for n-1 times **do**
    $W \leftarrow Tweak(Copy(S))$
    if $(Quality(W) > Quality(R)$ **then**
      $R \leftarrow W$
  $S \leftarrow R$
  if $(Quality(S) > Quality(Best)$ **then**
    $Best \leftarrow S$
until $Best$ is optimum or out of time
return $Best$

Problems with Hill Climbing

- Local optimum: usually won’t find global optimum
- Plateaus: algorithm gets stuck



How to Realize the Operations?

- Find a suitable representation of a candidate solution
  - Vector of numbers, list or set of objects, a tree, a graph, etc.
  - Representation must allow for implementing the operations for *Initialization*, *Tweak*, *Copy*, and *Quality*
- Example: fixed-length vector of real numbers as candidate solution
- Initialization operation:

$min \leftarrow$ minimum desired vector element value
$max \leftarrow$ maximum desired vector element value
$\vec{x} \leftarrow$ a new vector $\langle x_1, \dots, x_l \rangle$
for i from 1 to l **do**
  $x_i \leftarrow$ random number taken uniformly between min and max
return $\vec{x}$

How to Realize the Operations? (Cont.)

- Idea of Tweak operation:
  - Add random noise as small value to each number in the vector
  - But only for a given probability (often, we set $p \leftarrow 1$)

$\vec{x} \leftarrow$ vector $\langle x_1, \dots, x_l \rangle$ to be convolved
$p \leftarrow$ probability of adding noise to an element in the vector
$r \leftarrow$ half range of uniform noise
$min \leftarrow$ minimum desired vector element value
$max \leftarrow$ maximum desired vector element value

for i from 1 to l **do**
  if $p \geq$ random number chosen uniformly from 0.0 to 1.0 **then**
    repeat
      $n \leftarrow$ random number chosen uniformly from $-r$ to r *inclusive*
    until $min \leq x_i + n \leq max$
    $x_i \leftarrow x_i + n$
return $\vec{x}$

Exploration vs. Exploitation

- Exploration:
  - Explore the search space and avoid being trapped in a local maximum (very fast to find a locally good solution)
- Exploitation:
  - Exploiting local information to reliably move to (local) maximum (very important if the search space has many local optima)

- How to balance or even manipulate both aspects?
  - Parameter r allows as to tweak exploration vs. exploitation
  - Small r will fully exploit the locality to reach the local optimum
  - Large r will result in bounces through the search space (random search in the extreme case)

Single-State Global Optimization Algorithms

About Global Optimization

- An algorithm is guaranteed to find the global optimum, at least in theory
  - Often requires running the algorithm an infinite amount of time
  - Realized by having a chance to visit every possible solution in the solution space

- Why are the aforementioned approaches not global?
  - Tweak operation is bounded so that it stays in a local area

Random Search

- Concept: full explorative and no exploitation
- Idea: Randomly select a candidate solution

$Best \leftarrow$ random initial candidate solution
repeat
  $S \leftarrow$ a random candidate solution
  if $Quality(S) > Quality(Best)$ **then**
    $Best \leftarrow S$
until $Best$ *is optimum or out of time*
return $Best$

Global

Random Search | Hill Climbing with random restarts (global) | Hill Climbing with small r

Exploration | Exploitation

Local

Hill Climbing with Random Restarts

- Idea: Do Hill Climbing for some time and then start all over again from a different initial candidate solution

$T \leftarrow$ distribution of possible time intervals
$S \leftarrow$ random initial candidate solution
$Best \leftarrow S$
repeat
  $time \leftarrow$ random time in the near future, chosen from T
  repeat
    $S \leftarrow Tweak(Copy(S))$
    if $Quality(R) > Quality(S)$ **then**
      $S \leftarrow R$
  until *S* is optimum or *time* is up or out of time
  if $Quality(S) > Quality(Best)$ **then**
      $Best \leftarrow S$
  $S \leftarrow$ some random candidate solution
until $Best$ is optimum or out of time
return $Best$

Best Practices I

- Adjust the **modification procedure**
  - Tweak makes large, random changes
  - Global, because if long running, randomness will cause Tweak to try every solution
  - The more large, random changes, the more exploration
- Adjust the **selection procedure**
  - Change the algorithm so that you go *down*hills at least some time
  - Global, because if long running, you'll go down enough hills so that you can go up again at the global optimum hill
  - The more often going down hills, the more exploration

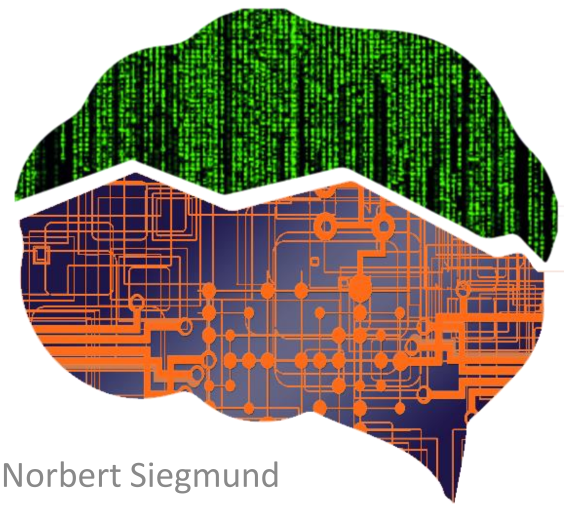
Best Practices II

- Jump to something new
  - Start from a new location every once in a while
  - Global, because if trying enough new locations, the optimum hill will be visited
  - The more frequent restarts, the more exploration
- Use a large sample
  - Try many candidate solutions in parallel
  - Global, because if enough parallel candidate solutions, one of them will be the optimum hill
  - More parallel candidate solutions, the more exploration

Currently: Single state optimization -> very small sample

Machine Learning for Software Engineering

Single-State Meta-Heuristics



Prof. Dr.-Ing. Norbert Siegmund
Intelligent Software Systems

Bauhaus-Universität Weimar

Recap

- What are heuristics and why do we need meta-heuristics for finding optimal solutions to a problem?
  - Standard approaches, such as gradient ascent do not work when function to be optimized is unknown
  - Scaling issues arise if search space is too large
  - We need heuristics that tell us how to search in an unknown search space

- What is the difference between exploration and exploitation?
  - Exploration aims at finding the global optimum by making random jumps throughout the whole search space
  - Exploitation aims at finding a local optimum (might be the global optimum) in sampling the local gradient using neighbor solutions

Recap II

- What is the relationship between exploration and exploitation and local search and global search?
- What is a local and what is a global search algorithm?
  - Local: Hill climbing
  - Global: Random search / walk

- How do they work?
- What are the essential parts?
  - Initialization procedure, assessment procedure, modification procedure, and selection procedure

Adjusting Modification Procedure: (1+1), (1+λ), (1,λ)

- Goal: Tweak operation *tending* toward small tweaks with *occasionally* large tweaks and can potentially make *any* possibly change

- Idea: Use Gaussian/Normal distributions as noise overlaid to the numbers in the vector representing a candidate solution
  - Most changes are close to zero, but some changes are huge
  - This is called **Gaussian Convolution**

(1+1) = Hill Climbing + Gaussian Convolution

Modification procedure:

$\vec{x} \leftarrow$ vector $\langle \mathrm{x}_1, \dots, \mathrm{x}_l \rangle$ to be convolved
$p \leftarrow$ probability of adding noise to an element in the vector
$\sigma^2 \leftarrow$ variance of the Normal distribution
$min \leftarrow$ minimum desired vector element value
$max \leftarrow$ maximum desired vector element value
for i from 1 to l $\boldsymbol{do}$
  if $p \geq$ random number chosen uniformly from 0.0 to 1.0 **then**
    $\boldsymbol{repeat}$
      $n \leftarrow$ random number chosen from the Normal distribution $\mathrm{N}\,(0, \sigma^2)$
    until $min \leq x_i + n \leq max$
    $x_i \leftarrow x_i + n$
return $\vec{x}$

Continued

- $(1+\lambda)$ = Steepest Ascent Hill Climbing + Gaussian Convolution
- $(1,\lambda)$ = Steepest Ascent Hill Climbing with Replacement + Gaussian Convolution

- Knobs we get via Gaussian Convolution:
  - σ^2 adjusting exploration vs. exploitation
  - Interacting with parameter n (number of parallel candidate solutions) of $(1,\lambda)$
    - If σ^2 large, we have noisy candidate solutions and look into many different solutions
    - If n is high simultaneously, the algorithm wipes out the poor candidates of such solutions aggressively
    - In this case, n is pushing toward exploitation whereas σ^2 toward exploration

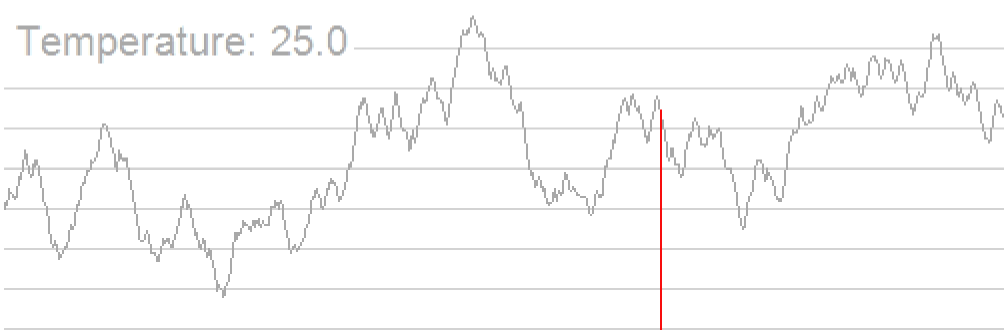
Simulated Annealing

Origin of Simulated Annealing (SA)

- SA is a *probabilistic* technique for *approximating* a *global* optimum
- Origin:
  - 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
  - For obtaining low energy states of a solid metal
- Approach:
  - Temperature of a solid metal is increased till it melts
  - Metal is cooled until its crystals are rearranged
  - Physical properties have changed

What is *Simulated* Annealing?

- Simulation of the annealing process
  - Solution to a combinatorial problem ~ states in a physical system
  - Cost of a solution ~ energy of a state
- Difference to Hill Climbing in its decision on when to replace S (original candidate solution) with R (newly tweaked one)
  - Avoid local optima by jumping randomly to a new state
  - Decrease the probability of jumping to a new state over time



By Kingpin13 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25010763

When to Replace a Candidate Solution?

- Three rules:
  - If a neighbor solution is better, always switch
  - If not, check the following:
    - How much worse are the neighboring solutions?
    - How high is the temperature of our system?
  - If the temperature is high -> more likely to switch to a worse solution
- Probability of switching the state:

$$P = e^{\left(\frac{currentEnergy - neighborEnergy}{temperature}\right)}$$

- Switch: $P >$ random number chosen uniformly 0.0 to 1.0

Transferred to our Problem

- Probability of switch is a function

$$P(t,R,S) = e^{\frac{Quality(R) - Quality(S)}{t}}$$

  - Where $t \geq 0$

Fraction is negative, because R is worse than S

If $Quality(R)$ << $Quality(S)$ or if t close to 0 -> P goes to 0

$t \leftarrow$ temperature with an initial high number
$S \leftarrow$ random initial *solution*
$Best \leftarrow S$
repeat
  $R \leftarrow Tweak(Copy(S))$
  if $(Quality(\mathrm{R}) > Quality(S)$ or if random nb btw. 0 to 1 $< e^{\frac{Quality(R)-Quality(S)}{t}}$ **then**
    $S \leftarrow R$
  Decrease t
  if $(Quality(S) > Quality(Best))$ **then**
    $Best \leftarrow S$
until *Best* is optimum or out of time or $t \leq 0$
return $Best$

Parameters

- What is a good starting temperature and how to reduce it?
  - Should be as big as biggest distance
  - Keeping t a long time high, we do more exploration
  - Reduce with: $t_{new} = \alpha t_{current}\ with\ \alpha < 1$
- Shall I reduce the neighborhood to avoid jumping to heavily around?
  - Yes! See Adaptive Simulated Annealing

In general: Experience & Experimentation
All values are problem dependent and there is no silver bullet

Adaptive Simulated Annealing (ASA)

- Algorithm controls temperature schedule and random step selection based on the algorithm’s progress
- Idea: temperature is not reduced if there is little progress on the quality

- Many sophisticated adaptations possible

Tabu Search

- Idea: Keep a list (the **tabu list L**) of already visited candidate solutions and refuse to visit them again until some time has gone
- In essence, we wander up hill, do not stay there (as this is not prohibited) and wander down the other side of the hill
- List is implemented as priority queue (if maximum capacity of L is reach, the oldest element will be removed)

- Realized by adapting Steepest Ascent with Replacement

Tabu Search Algorithm

$l \leftarrow$ Destired maximum tabu list length
$n \leftarrow$ number of tweaks desired to sample the gradient
$S \leftarrow$ random initial *solution*
$Best \leftarrow S$
$L \leftarrow \{ \quad \}$ a tabu list of maximum length l
repeat
  if $Length(L) > l$ **then**
    Remove oldest element from L
  $R \leftarrow Tweak(Copy(S))$
  for n-1 times **do**
    $W \leftarrow Tweak(Copy(S))$
    if $W \notin L$ and $(Quality(W) > Quality(R)$ or $R \in L)$ **then**
      $R \leftarrow W$
  if $R \notin L$ and $Quality(R) > Quality(S)$ **then**
    $S \leftarrow R$
    Enqueue R into L
  if $(Quality(S) > Quality(Best))$ **then**
    $Best \leftarrow S$
until *Best* is optimum or out of time
return $Best$

Limitations

- Works only in discrete spaces
  - If applied to real-valued spaces, we need to refuse “similar” solutions that are already in the tabu list
- When search space has many dimensions, it still will stay at the same hill (easy to find a nonvisited neighbor)
  - Instead of saving the candidate solutions, we might save the changes we have made to a candidate solution
  - For ex. save deleting and adding edges in the TSP scenario
  - Result: Feature-Based Tabu Search

Iterated Local Search (ILS)

- Improved version of Hill Climbing with Random Restarts
- Idea: Restart at a position where it likely finds a new local optimum
  - Tries to search the space of local optima
  - Approach: Find a local optimum, then searches for a nearby local optimum, and so on
- Heuristic: Find better local optimum in the neighborhood of your current local optimum (better than complete random)
  - Restart positions not entirely random, but random in a certain distance to a “home base” local optimum
  - If a new local optimum has been found, decide whether it becomes the new “home base”

ILS Algorithm

$T \leftarrow distribution\ of\ possible\ time\ intervals$
$S \leftarrow$ random initial *solution*
$Best \leftarrow S$
$H \leftarrow S$ (the current home base)
repeat
  $time \leftarrow$ random time in the near future chosen from T
    repeat
      $R \leftarrow Tweak(Copy(S))$
      if $(Quality(R) > Quality(S))$ **then**
        $S \leftarrow R$
    until *S* is optimum or *time* is up or out of time
  if $(Quality(S) > Quality(Best))$ **then**
    $Best \leftarrow S$
  $H \leftarrow NewHomeBase(H, S)$
  $S \leftarrow Perturb(H)$
until *Best* is optimum or out of time
return $Best$

Decides whether to change the home base

Difficult to tune

Make a large Tweak to search farther away from the home base

Take Home Message:

- Many approaches possible to tune the search between local optimization by exploiting the gradient of neighbor solutions and global optimization by exploring the whole configuration space using random jumps
- Which approach and how to balance exploration and exploitation is problem dependent
- Start with a good encoding of the problem and then try out some techniques and probably adjust some settings

Next Lecture & Literature

- Multi-State optimization algorithms (population methods)
  - Evolution strategies
  - Genetic algorithms
  - Differential Evolution