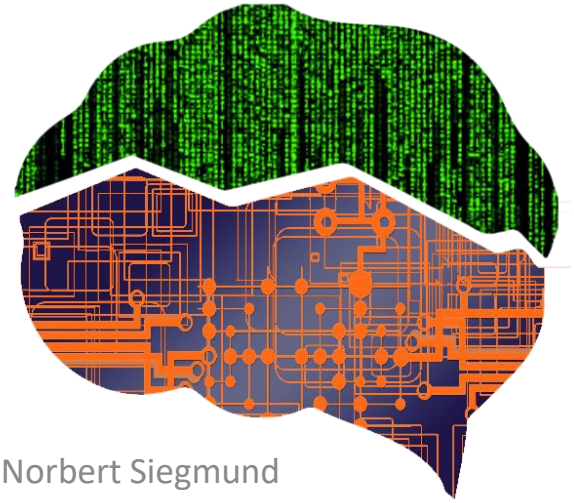
Search-Based Software Engineering

Single-State and Multi-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 $\langle x_1, \ldots, x_l \rangle$ 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 ***do***
  if $p \geq$ random number chosen uniformly from 0.0 to 1.0 **then**
    repeat
      $n \leftarrow$ random number chosen from the Normal distribution $N(0, \sigma^2)$
    until $min \leq x_i + n \leq max$
    $x_i \leftarrow x_i + n$
return $\vec{x}$

Continued

- (1+λ) = Steepest Ascent Hill Climbing + Gaussian Convolution
- (1,λ) = 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,λ)
    - 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

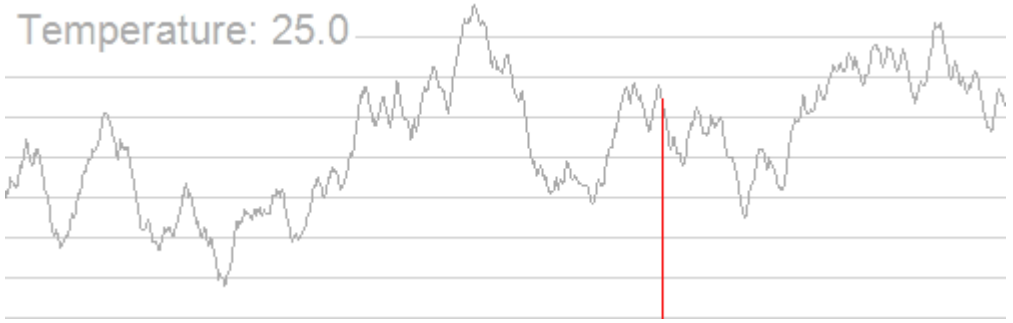
Simulated Annealing

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

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

Fraction is negative, because R is worse than S

  - Where $t \geq 0$

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 ($Quality$(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

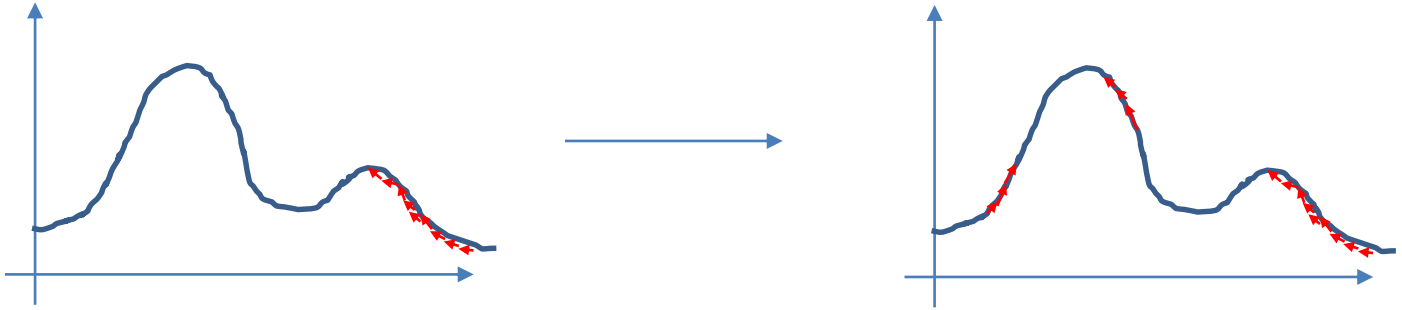
Single-State: Take Home Message

- 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

Multi-State Optimization (Population Methods)

What is new?

- Instead of saving the globally best solution or the currently best solution, we keep a *sample* of candidate solutions



- Difference to parallel hill climbing:
  - Candidate solutions affect how other candidates will climb the hill
  - For ex. good solutions will replace bad ones by new solutions
  - For ex. bad solutions will be tweaked in the direction of good ones

Evolutionary Algorithms (EAs)

- Idea: borrow concepts from biology regarding genetics, evolution, and reproduction to resample the current candidate solutions
  - New candidate solutions are created or existing ones revised based on the results of older solutions

- General process:
  - Construct initial population
  - Iterate over:
    - **Assess fitness** of all individuals in the population
    - Use fitness information to **breed** a new population
    - **Join** the parents and children in some fashion to form the next-generation population

Terms & Notation

Term	Description
Individual	Candidate solution
Child and parent	Child is tweaked copy of a candidate solution (parent)
Population	Set of candidate solutions
Fitness	Quality
Fitness landscape	Quality function (usually unknown)
Selection	Select an individual based on the fitness value
Mutation	Tweak operation
Recombination & Crossover	Tweak operation with two parents as input and doing some operations over the elements of the parents to produce two children
Genotype or genome	Data structure of an individual
Chromosome	Genotype of a fixed-length vector
Gene	A specific slot or element in a chromosome
Allele	Particular setting of a gene
Phenotype	How the individual operates during fitness assessment
Generation	One cycle of fitness assessment, breeding, and population generation; or the population produced at each cycle

Breed

Basic Algorithm

$P \leftarrow$ build initial population
$Best \leftarrow empty$
repeat
  $AssessFitness(P)$
  for each individual $P_i \in P$ **do**
    if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
      $Best \leftarrow P_i$
  $P \leftarrow Join(P, Breed(P))$
until $Best$ is optimum or out of time
return $Best$

First, we need to construct a set of (random) candidate solutions

Second, compute the quality of each candidate solution and store it

Third, breed new solutions based on the quality of each candidate solution

Forth, join the newly bred candidate solutions with the solutions of the current population

Difference to single-state algorithms: We need to assess the fitness (quality) of all candidate solutions before we can decide which one survives/to breed

From Basic to Concrete Algorithms

- Breed operation:
  - How to select parents from the old population and how to tweak them to make children?
- Join operation:
  - Shall we replace the parent population completely or keep some of them?
- Initialization operation:
  - If you don't know anything about the "good-solution-area" -> random
  - If you have knowledge, bias the random generation toward the "good-solution-area" (e.g., include / seed user-defined solutions in the initial population)
  - Make sure that you use only unique individuals

Evolution Strategies (ES)

- Invented in mid 1960s by

Ingo Rechenberg

Hans-Paul Schwefel

- Characteristics:
  - Selecting individuals using Truncate Selection
  - Only use mutation as tweak realization
- Simplest algorithm is (μ,λ)
  - λ is the number of individuals, randomly generated
  - Delete from the population all, but μ fittest individuals
  - Each of the fittest individuals produce λ/μ children (mutation)
  - Join operation replaces the parents by the children

(μ,λ) Algorithm

$\mu \leftarrow$ number of parents that are used to breed children
$\lambda \leftarrow$ number of children to be generated by the parents
$P \leftarrow \{\ \ \}$
for λ times **do**
  $P \leftarrow P\ \cup \{$random individual$\}$
$Best \leftarrow empty$
repeat
  for each individual $P_i \in P$ **do**
    $AssessFitness(P_i)$
    if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
      $Best \leftarrow P_i$
  $Q \leftarrow$ the μ individuals in P whose $Fitness$ are greatest
  $P \leftarrow \{\}$
  for each individual $Q_i \in Q$ **do**
    for λ / μ times **do**
      $P \leftarrow P\ \cup \{Mutate(Copy(Q_i))\}$
until $Best$ is optimum or out of time
return $Best$

Exploration vs. Exploitation in (μ,λ)

- λ controls sample size for each population
  - Equal to *n* in Steepest-Ascent Hill Climbing with Replacement
  - If it goes to infinity, it is random search
- μ controls the selectivity of the algorithm
  - If it is low, it maximizes exploitation
- The degree of mutation
  - Amount of noise that is used to mutate an individual to produce a new child
  - High noise means explorative and low means exploitative

(μ+λ) Algorithm

- The only difference is the join operation

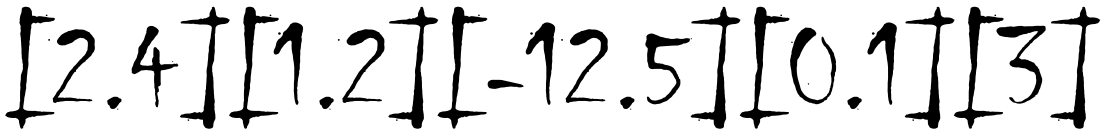
(μ,λ)	(μ+λ)
$P \leftarrow \{\}$	$P \leftarrow \{Q\}$

- The fittest parents survive and compete with their children in the next generation

- Can cause premature convergence as the parents restrict exploration
- Compare: Steepest Ascent Hill Climbing (1+λ) with Steepest Ascent Hill Climbing with Replacement (1, λ)
  - So, (μ+λ) ES is the more general algorithm

Realizing Mutations for ES

- Usually, the individual is represented as fixed-length vector of real numbers



- Numbers are generated and mutated with Gaussian Convolution (see last lecture)
  - Normal distribution with a given variance: σ^2 = mutation rate
  - Variance controls exploration vs. exploitation
- How to set σ^2?
  - Static: with our without domain knowledge
  - Adaptive: changing σ^2 over time = adaptive mutation rate

Adaptive Mutation Rate: One-Fifth Rule

- If more than $\frac{1}{5}$ children are fitter than their parents, too much exploitation -> increase σ^2
- If less than $\frac{1}{5}$ children are fitter than their parents, too much exploration -> decrease σ^2
- If exactly $\frac{1}{5}$ children are fitter than their parents, keep σ^2

Evolutionary Programming (EP) is very similar to ES, but often broader than ES with respect to the representation of an individual (and so the mutation operation is different)

Take Home Message

- Single-State meta-heuristics
  - Adjust exploration and exploitation
  - Reduce exploration when progressing toward the optimum
  - Use various approaches (e.g. Tabu Search or ILS) to leave a local optimum

- Multi-State meta-heuristics
  - The change to a candidate solution will be made depending on the current status (i.e. fitness value) of other individuals
  - There is an information flow between the individuals
  - New solutions are generated based on mutations and combinations of old solutions