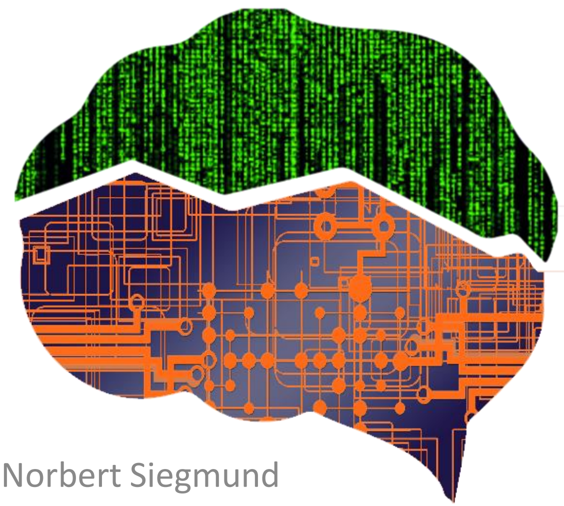
Search-Based Software Engineering

Exploitative Multi-State Meta-Heuristics



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

Bauhaus-Universität Weimar

Recap I

- Important operations to find optimal solutions are crossover, individual selection, and mutation
- Individual selection:
  - Aims at selecting those individuals for the crossover operation that provide the best fitness values
  - (Stochastic) Proportionate Selection, Tournament Selection
- Crossover:
  - 1-point and 2-point cut regions out of an individual and cross this region with another one
  - Uniform crossover crosses randomly chosen genes
  - Crossover alone is not sufficient: Line Recombination

Recap II

- Mutation:
  - Gaussian Convolution for floating-point values
  - Adaptive mutation, e.g., using the one-fifth-rule

- Evolutionary strategies vs. genetic algorithm
  - ES use only mutation, which has its limits (hyper cube)
  - GA uses also crossover and is most often used with a fixed-length binary vector representation

Exploitative Variations of Population-Based Optimization Techniques

Elitism as a General Method

- Simple adaptation of GA: Insert the fittest individuals of the current generation into the next generation: the elites
- Very similar to μ+λ algorithm with same pros and cons (e.g., premature convergence)

$popsize \leftarrow$ desired population size
$n \leftarrow$ number of elite individuals
$P \leftarrow \{\ \}$
for $popsize$ 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 n fittest individuals in $P\}$
  for $\frac{popsize - n}{2}$ times **do**
    $Parent\ P_a \leftarrow SelectWithReplacement(P)$
    $Parent\ P_b \leftarrow SelectWithReplacement(P)$
    $Children\ C_a, C_b \leftarrow Crossover(Copy(P_a), Copy(P_b))$
    $Q \leftarrow Q \cup \{Mutate(C_a), Mutate(C_b)\}$
  $P \leftarrow Q$
until $Best$ is optimum or out of time
return $Best$

Steady-State Genetic Algorithm

- Alternative to the common generational GAs in the way that they do not update/replace the whole generation, but do it piecewise

$popsize \leftarrow$ desired population size
$P \leftarrow \{\quad\}$
for $popsize$ times **do**
  $P \leftarrow P \cup \{$random individual$\}$
$Best \leftarrow empty$
for each individual $P_i \in P$ **do**
  $AssessFitness(P_i)$
  if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
    $Best \leftarrow P_i$

Initialization

Steady-State GA II

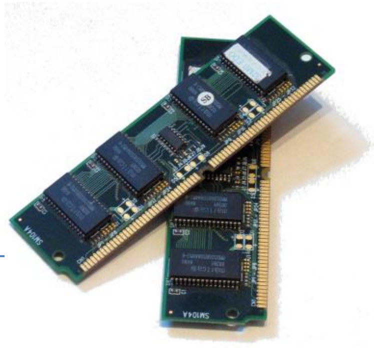
repeat

$Parent\ P_a \leftarrow SelectWithReplacemnt(P)$
$Parent\ P_b \leftarrow SelectWithReplacemnt(P)$
$Children\ C_a, C_b \leftarrow Crossover\big(Copy(P_a), Copy(P_b)\big)$
$C_a \leftarrow Mutate(C_a)$
$C_b \leftarrow Mutate(C_b)$
$AssessFitness(C_a)$
If $Fitness(C_a) > Fitness(Best)$ **then**
$Best \leftarrow C_a$
If $Fitness(C_b) > Fitness(Best)$ **then**
$Best \leftarrow C_b$
$Individual\ P_d \leftarrow SelectForDeath(P)$
$Individual\ P_e \leftarrow SelectForDeath(P)$
$P \leftarrow P - \{P_d, P_e\}$
$P \leftarrow P \cup \{C_a, C_b\}$

until $Best$ is optimum or out of time
return $Best$

Note that only two parents will be selected for breeding new children and only two individuals will be removed in the whole generation step

Discussion



- Benefits:
  - Requires only half memory (since only one population is maintained at a time)
  - Exploitative, because parents stay in the generation as long as they are not the worst individuals
- Drawbacks:
  - Premature convergence depending on *SelectForDeath* operation (removing only unfit individuals might remove explorative individuals -> we stay at a local optimum)
- Further knobs:
  - Replace not two, but n individuals; replace at random
  - Make mutation and crossover noise, etc.

Tree-Style Genetic Programming Pipeline

- What is genetic programming?
  - Research area of using meta-heuristics in finding an optimal program
- Common representation for a genetic programming problem is a tree (more on representations in the exercise)

- How to do the tweak operation?
  - With 0.9 probability do a crossover, with 0.1 probability copy the parents
  - No mutation operation (ie., not global)
  - Tournament selection with t=7

Hybrid Optimization Algorithms

- For ex. combine evolutionary algorithm with hill climbing
  - EA in the inner loop and hill climbing in the outer loop
  - Realized as extension to Iterated Local Search
- Or, use EA for exploration (outer loop) and local optimization as inner loop as exploitation
  - Implement hill climbing during the fitness assessment phase to revise and improve each individual at the time it's assessed
  - Revised individuals replace original ones

Hybrid Algorithm ES+HC

$t \leftarrow$ number of iterations for hill climbing ← Adjusts exploitation
$P \leftarrow \{ \quad \}$
$Best \leftarrow empty$
repeat
  $AssessFitness(P)$
  for each individual $P_i \in P$ **do**
    $P_i \leftarrow HillClimb(P_i)$ for t iterations
    if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
      $Best \leftarrow P_i$
  $P \leftarrow Join(P, Breed(P))$
until $Best$ is optimal or out of time
return $Best$

Other examples for combining global optimization with local refinement:

- Iterated local search (hill climbing inside more explorative hill climbing)
- Hill climbing with random restarts

Memetic Algorithm

Global optimization

Local optimization (could be problem-specific):

- Meta-heuristic,
- Heuristic
- Machine learning

- Idea: Individuals improve their self during fitness assessment and pass along this improvement to their offspring

Jean-Baptiste Lamarck
(wrong evolution theory)

Memetic Algorithm – Pseudo Code

$t \leftarrow$ number of iterations for local improvement
$p \leftarrow$ probability of performing local improvement
$P \leftarrow$ {initial population}
$Best \leftarrow empty$
repeat
  $AssessFitness(P)$
  $P \leftarrow Mutate\big(Copy(P)\big)$
  $W \leftarrow selectSubsetForLocalImprovement(P)$
  for each individual $W_i \in W$ **do**
    if random number between 0 and $1 < p$ **then**
      Perform local improvement of W_i for t times
  $P \leftarrow Join(P, Breed(W))$
until $Best$ is optimal or out of time
return $Best$

Further Hybrid Ideas

- Learnable Evolution Model
  - Alternate between evolution and machine-learning classification
- Meta-heuristics optimize tuning parameters of other meta-heuristics (Meta-Optimization and Hyperheuristics)
  - E.g., use GA to tune optimal mutation rate, crossover operation, etc. for a second GA, working on the actual problem

Scatter Search

- Combination of evolutionary algorithm with hill climbing, line recombination, (μ+λ), and explicit injection of individuals for exploration
  - Combines several exploitative techniques
  - Enforces diversity of individuals
- Approach
  - Start with initially seeded individuals
  - Next, production of a large number of random individuals that are very different from one another and the seeds
  - Next, hill climbing on each of these individuals
  - Next loop:
    - Truncate population to a small size consisting of some very fit individuals and some very diverse individuals
    - On this small population, do line recombination (crossover) + mutation
    - Next, do hill climbing on these produced individuals and add them to the population; proceed with the loop

How to Produce Diverse Individuals?

- Distance measure to rate similarity of individuals
  - E.g. for real-valued vectors $\vec{v}, \vec{u}$: $\sqrt{\sum_i (v_i - u_i)^2}$ use the Euclidean distance
  - Diversity of P_i is $\sum_j distance(P_i, P_j)$
- Rank the individuals based on their diversity and select the most diverse individuals
- Or, use tournament selection with diversity to the seed as size parameter

Algorithm I (initial setup)

$Seeds \leftarrow$ initial collection of individuals provided by the user
$initsize \leftarrow$ initial sample size
$t \leftarrow$ number of iterations for hill climbing
$n \leftarrow$ number of individuals to be selected based on fitness
$m \leftarrow$ number of individuals to be selected based on diversity
$P \leftarrow Seeds$
for $initsize - \left||Seeds|\right|$ times **do**
  $P \leftarrow P \cup \{ProduceDiverseIndividual(P)\}$ ← Inject new individuals to the first population based on diversity measure
$Best \leftarrow empty$
for each individual $P_i \in P$ **do**
  $P_i \leftarrow HillClimb(P_i)$ for t iterations
  $AssessFitness(P_i)$
  if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
    $Best \leftarrow P_i$

Do hill climbing on each, so that we have a highly tuned starting set of individuals

Algorithm II (optimization loop)

repeat
$B \leftarrow$ the fittest n individuals in P
$D \leftarrow$ the most diverse individuals in P
$P \leftarrow B \cup D$
$Q \leftarrow \{\,\}$
for each individual $P_i \in P$ **do**
for each individual $P_j \in P$ where $i \neq j$ **do**
$Children\ C_a, C_b \leftarrow Crossover\left(Copy(P_i), Copy(P_j)\right)$

$C_a \leftarrow Mutate(C_a)$	$C_b \leftarrow Mutate(C_b)$
$C_a \leftarrow HillClimb(C_a)$ for t iterations	$C_b \leftarrow HillClimb(C_b)$ for t iterations
$AssessFitness(C_a)$	$AssessFitness(C_b)$
if $Fitness(C_a) > Fitness(Best)$ **then**	**if** $Fitness(C_b) > Fitness(Best)$ **then**
$Best \leftarrow C_a$	$Best \leftarrow C_b$

$Q \leftarrow Q \cup \{C_a, C_b\}$
$P \leftarrow Q \cup P$
until $Best$ is optimal or out of time
return $Best$

Store the best individuals seen so far (for exploitation)

Store the most diverse individuals seen so far (for exploration)

Use Line Recombination here

Mutate and hill climb each child individually and check whether we found the best solution so far

Differential Evolution

- Adaptive mutation algorithm
  - Specifies the size of mutations based on the current variance in the population
  - If population is wide spread (diverse), mutate operation will make large changes
  - If population is condensed in a certain region, mutate operation will make only small changes
- Works only for metric-based vector spaces

Idea of Differential Evolution

- For each individual $\vec{\imath}$ in a population generate a child as follows:
  - Select three additional individuals $\vec{a}, \vec{b}, \vec{c}$ at random
  - Subtract the two vectors to get their distance $\vec{d} = \vec{b} - \vec{c}$
  - Add this distance vector to the individual: $\vec{a} \leftarrow \vec{a} + \vec{d}$
  - Do crossover of $\vec{\imath}$ with $\vec{a}$ to construct the child
- Build a group of children this way and replace a child with its parent if it has a better fitness score

- At the beginning, we are usually spread throughout the solutions space and do more exploration
- Later, we will converge to a smaller region and want then only small mutations
- Selection procedure is different here: first select random individuals and produce children, then do the selection—**survival selection**—(before it was first selection, then breeding—**parent selection**—)

Differential Evolution Algorithm

$\alpha \leftarrow$ mutation rate
$popsize \leftarrow$ desired population size
$P \leftarrow \langle\,\rangle$ empty vector of $popsize$ size
$Q \leftarrow \langle\,\rangle$
for i from 1 to $popsize$ **do**
  $P_i \leftarrow new\ random\ individual$
$Best \leftarrow empty$

Initialization

repeat
  for each individual $P_i \in P$ **do**
    $AssessFitness(P_i)$
    if $Q \neq empty$ and $Fitness(Q_i) > Fitness(P_i)$ **then**
      $P_i \leftarrow Q_i$
    if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
    $Best \leftarrow P_i$
  $Q \leftarrow P$

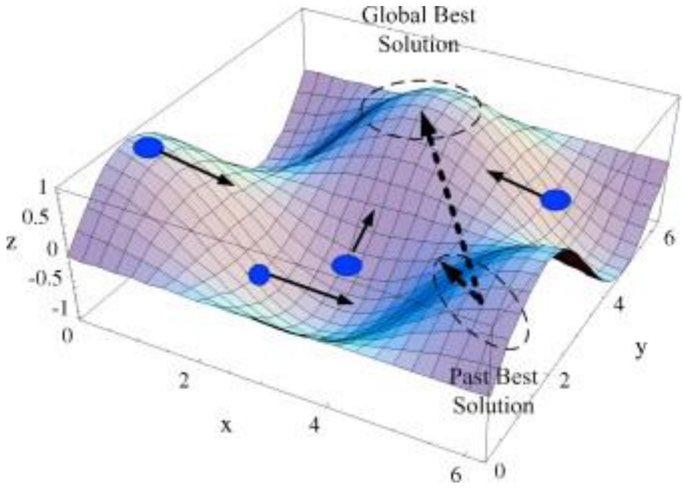
Keep a temporary population of the best individuals

  for each individual $Q_i \in Q$ **do**
    $\vec{a} \leftarrow$ a copy of an individual other than Q_i chosen randomly with replacement from Q
    $\vec{b} \leftarrow$ a copy of an individual other than Q_i or $\vec{a}$ chosen rand. with replacement from Q
    $\vec{c} \leftarrow$ a copy of an individual other than $Q_i, \vec{a}$, or $\vec{b}$ chosen rand. with replacement from Q
    $\vec{d} \leftarrow \vec{a} + \alpha * (\vec{b} - \vec{c})$
    $P_i \leftarrow$ one child from $Crossover(\vec{d}, Copy(Q_i))$
until $Best$ is optimal or out of time
return $Best$

Breed a child per parent, based on distances to other individuals in the population

Particle Swarm Optimization (PSO)

- Stochastic optimization technique
- Idea inspired by swarm behavior (flocks) of animals
- Key difference: PSO has no selection operation (no resampling of the population)



PSO Approach

- Static population of individuals that are tweaked depending on new discoveries in the search space
  - Resembles directed mutation toward promising areas (i.e., best found solutions so far)
  - Works usually on real-valued genes
  - Requires a metric space (vs. eg., mutating a tree or graph toward a certain region)
- Assumes information exchange (social interactions) among the individuals
  - Keep track of *global best*, *regional best*, and *self best* solution
- Usually referred to as **swarm** of **particles** instead of population of individuals

Particles

- Location:
  - Vector in space $\vec{x} = \langle x_1, x_2, \dots \rangle$
  - Same as genotype in ES
- Velocity:
  - Speed and direction at which a particle will move in each step, encoded as a vector $\vec{v} = \langle v_1, v_2, \dots \rangle$
- Example:
  - At time t, $\vec{v} = \vec{x}^{(t)} - \vec{x}^{(t-1)}$

PSO Explained

- Initialization:
  - Each particle starts from a random position with a random velocity vector
  - Idea: select two random points in the space and use half of the distance as velocity vector
- Memorization (keep track of):
  - Local best location: $\vec{x}^*$ that has $\vec{x}$ discovered so far
  - Regional best location: $\vec{x}^+$ that any particle that exchanges information with $\vec{x}$ has discovered so far (eg., grid neighbors or in each iteration randomly chosen particles)
  - Global best location: $\vec{x}^!$ that any particle globally has found so far

PSO Iterations

- Each time step, do the following:
  - Assess fitness of each particle and update best-discovered locations
  - Determine how to Mutate:
    - For each particle $\vec{x}$, we update its velocity vector $\vec{v}$ by adding in (to some degree) a vector pointing towards $\vec{x}^*$ + random noise for each dimension separately
  - Mutate each particle by moving it along its velocity vector

PSO Initialization

$swarmsize \leftarrow$ desired population/swarm size
$\alpha \leftarrow$ proportion of velocity to be retained
$\beta \leftarrow$ proportion of personal best to be retained
$\gamma \leftarrow$ proportion of the informants' best to be retained
$\delta \leftarrow$ proportion of the global best to be retained
$\varepsilon \leftarrow$ jump size of a particle
$P \leftarrow \{\}$ empty *set*
for *swarmsize* times **do**
  $P \leftarrow P \cup$ {new random particle $\vec{x}$ with a random initial velocity $\vec{v}$}
$\overrightarrow{Best} \leftarrow empty$

Define the probabilities of keeping certain best positions

α: how much of the original velocity is retained
β: how much of the personal best is mixed in (large β moves particles more to their own best solution, rather than towards the global best -> lot of separate hill climbers and no joint searchers)
γ: how much of the informants' best is mixed in (in the middle of β and δ)
δ: how much of the global best is mixed in (large d moves particles more to the best known location -> leads to a single large hill climber and no separate hill climbers -> threatens exploitation, so commonly set to 0)
ε: how fast the particles move (large e leads to large jumps towards promising locations at the danger of overshooting; often set to 1)

PSO Algorithm

repeat
  for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do**
    $AssessFitness(\vec{x})$
    if $\overrightarrow{Best} == empty$ or $Fitness(\vec{x}) > Fitness(\overrightarrow{Best})$ **then**
      $\overrightarrow{Best} \leftarrow \vec{x}$
  for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do**
    $\overrightarrow{x^*} \leftarrow$ previous fittest location of $\vec{x}$
    $\overrightarrow{x^+} \leftarrow$ previous fittest location of informants of $\vec{x}$, including $\vec{x}$
    $\overrightarrow{x^!} \leftarrow$ previous fittest location of any particle (global best)
    for each dimension i **do**
      $b \leftarrow$ random number from 0.0 to β inclusive
      $c \leftarrow$ random number from 0.0 to γ inclusive
      $d \leftarrow$ random number from 0.0 to δ inclusive
      $v_i \leftarrow \alpha v_i + b(x_i^* - x_i) + c(x_i^+ - x_i) + d(x_i^! - x_i)$
  for each particle $\vec{x} \in P$ with velocity $\vec{v}$ **do**
    $\vec{x} \leftarrow \vec{x} + \varepsilon\vec{v}$
until $\overrightarrow{Best}$ is optimal or out of time
return $\overrightarrow{Best}$

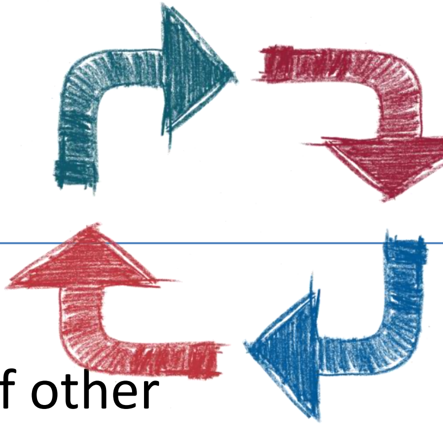
Update global best

Update best locations to prepare the according mutation

Stochastically update the velocity depending on the best locations

Update to new position

What Else Can We Do?



- **Coevolution**
  - Fitness of an individual depends on the presence of other individuals in the populations
  - So, individual A is superior to B if it depends on the presence of an individual C
  - Goal is to have *robust* solutions and solving complex, high-dimensional problems by dividing them

  - 1-Population Competitive Coevolution:
    - Fitness of individuals based on games they play against each other
  - 2-Population Competitive Coevolution:
    - Two subpopulations: Fitness of individual in pop 1 depends on how many individuals in pop 2 it can defeat in some game (& vice versa)
  - …

What Else Can We Do?

- **Parallelization** of Metaheuristics: 5 ways
  - Do separate runs in parallel
  - Do one run and split the fitness assessment task (+ other operations) among multiple threads on the machine
  - Do separate runs in parallel and synchronize from time to time the best individuals (i.e., island models)
  - Do one run and distribute the fitness assessment to remote machines (i.e., master-slave/client-server/ distributed fitness assessment)
  - Do one run with a selection procedure presuming that individuals are spread out in a parallel array on a vector computer (i.e., spatially embedded / fine-grained models)

Take Home Message:

- Exploit more with the elites of a generation
- Save memory using steady state GA
- Genetic programming is a discipline to generate programs using genetic algorithms
  - Tree-based representation as opposed to vector-based
- Hybrid optimization techniques
  - Combine EAs with hill climbing or machine learning (memetic algorithms)
  - Scatter search goes even beyond that in producing diverse individuals

Take Home Message:

- Differential evolution as adaptive mutation algorithm
  - Current variance of the population specifies the kind and strength of the mutation
  - Survival selection instead of parent selection

- Particle swarm optimization with no selection operation
  - Particles store position, velocity, and best positions
  - Particles move based on the velocity and neighbors’ best solutions

Next Lecture

- Multi-Objective Optimization
  - NSGA-II
  - Pareto Front