Artificial Intelligence

Evolutionary Algorithms

Lesson 2: Meta heuristics and related optimization techniques

Vincenzo Piuri

Università degli Studi di Milano

Contents

- Meta heuristic
- Local search methods
- Example: The Traveling Salesman Problem
- EA-related methods

Meta heuristic

What is a meta heuristic?

- Algorithmic method for approximately solving of a combinatorial optimization problem
- Defines abstract sequence of steps which can be applied on every kind of problem
- But: single steps has to be implemented in a problem-specific way

  $\Longrightarrow$ problem-specific heuristic

Application of meta heuristics

- On problems where no efficient solving algorithm is known
  - e.g. combinatorial optimization problems
- Finding an optimal solution is usually not guaranteed
- In comparison with optimal solution: good solutions can be arbitrarily bad
- Success and runtime depends on
  - problem definition
  - implementation of particular steps

  $\Longrightarrow$ EAs are meta heuristics, too

Local Search Methods

Local Search Methods (1)

- Given: optimization problem $(\Omega, f, \succ)$
- Desired: find element $x \in \Omega$, which optimizes f (max. or min.)
  - Without loss of generality: find an element $x \in \Omega$, which maximizes f (should f be minimized, then we consider $f' \equiv -f$)

  $\Longrightarrow$ Local search methods to find local optima in Ω
- Assumption: $f(x_1)$ and $f(x_2)$ differ slightly for similar $x_1, x_2 \in \Omega$ (no huge jumps in f)
- Applicable for arbitrary Ω to find local optima

Local Search Methods (2)

- Local search methods = special case of EA
- Population: 1 solution candidate $\Longrightarrow$ various consequences
- Recombination operator is not reasonable as there is only one individual
  - changes: mutation represent variation operator
  - selection: newly created individual instead of parental individual into next generation?

  $\Longrightarrow$ One fundamental algorithm for all local search methods
- Variants by different acceptance criterion
- Individuals contain usually no additional information $\mathcal{Z} = \emptyset$
- Genotype $\mathcal{G}$ depends on problem (as always)

Local Search Methods (3)

Algorithm 1 fundamental algorithm of local search

Input: objective function F
Output: solution candidate A

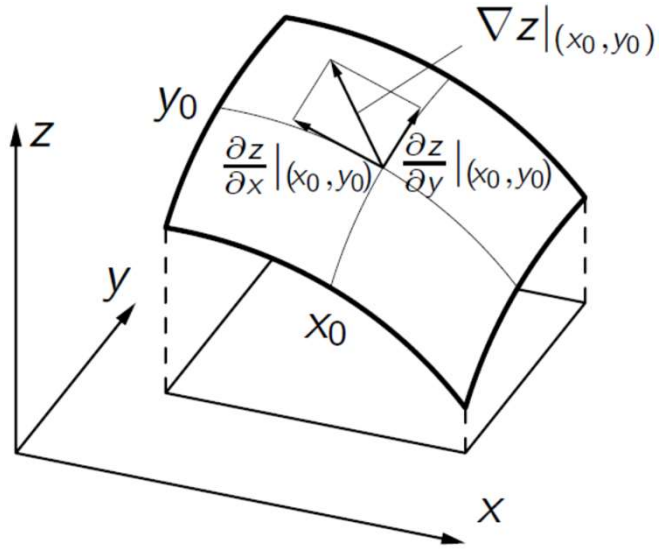
```
 1: t ← 0
 2: A(t) ← create solution candidate
 3: evaluate A(t) by F
 4: while termination criterion isn't fulfilled {
 5:     B = vary A(t)
 6:     evaluate B by F
 7:     t ← t + 1
 8:     if Acc(A(t − 1).F, B.F, t) {          /* acceptance criterion */
 9:         A(t) ← B
10:     } else {
11:         A(t) ← A(t − 1)
12:     }
13: }
14: return A(t)
```

- *Acc*: variably implemented on certain local search methods

Gradient Ascent or Descent (1)

- Assumption: $\Omega \subseteq \mathbb{R}^n$ and $f: \Omega \to \mathbb{R}$ is differentiable
- Gradient: differential operation that creates a vector field

  $\Longrightarrow$ computes vector into the direction of the steepest ascent of the function in a point



- Illustration of the gradient of $z = f(x, y)$ at point (x_0, y_0)

$$\nabla z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}|_{(x_0,y_0)}, \frac{\partial z}{\partial y}|_{(x_0,y_0)} \right)$$

Gradient Ascent or Descent (2)

- Idea - Start at a randomly chosen point, then make small steps in the search space Ω in (or against) the direction of the steepest slope of the function until a (local) optimum is reached

  1) Choose a (random) starting point $x^{(0)} = \left(x_1^{(0)}, \ldots, x_n^{(0)}\right)$

  2) Compute the gradient at the current point $\boldsymbol{x}^{(t)}$

$$\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(t)}\right) = \left(\frac{\partial}{\partial x_1} f\left(\boldsymbol{x}^{(t)}\right), \ldots, \frac{\partial}{\partial x_n} f\left(\boldsymbol{x}^{(t)}\right)\right)$$

  3) Make a small step in the direction of the gradient

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \eta \, \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(t)}\right)$$

  η: step width parameter ("learning rate" in ANN)

  4) Repeat steps 2 and 3 until some termination criterion is fulfilled (e.g. user-specified number of steps has been executed, gradient is smaller than a user-specified threshold)

Gradient Ascent or Descent: Problems

- Choice of the step width parameter
  - Too small value $\Longrightarrow$ large runtime until optimum is reached
  - Too large value $\Longrightarrow$ oscillations, jump back and forth in Ω
  - Solution: momentum term, adaptive step width parameter

- Getting stuck in local maxima
  - Due to local gradient information, maybe only local maxima is reachable
  - Problem cannot be remedied in general
  - Chance improvement: multiple execution with different starting points

Hill Climbing (1)

- Idea – If f is not differentiable, determine direction in which f increases by evaluating random points in the vicinity of the current point
  1) Choose a (random) starting point $x_0 \in \Omega$
  2) Choose a point $x' \in \Omega$ "in the vicinity" of x_t (e.g. by a small random variation of x_t)
  3) Set

$$x_{t+1} = \begin{cases} x' & \text{if } f(x') > f(x_t), \\ x_t & \text{otherwise} \end{cases}$$

  4) Repeat steps 2 and 3 until a termination criterion is fulfilled

Hill Climbing (2)

- Pseudocode of acceptance criterion in fundamental algorithm

Algorithm 2 Acceptance criterion of Hill Climbing

Input: fitness of parents $A.F$, fitness of offspring $B.F$, generation t

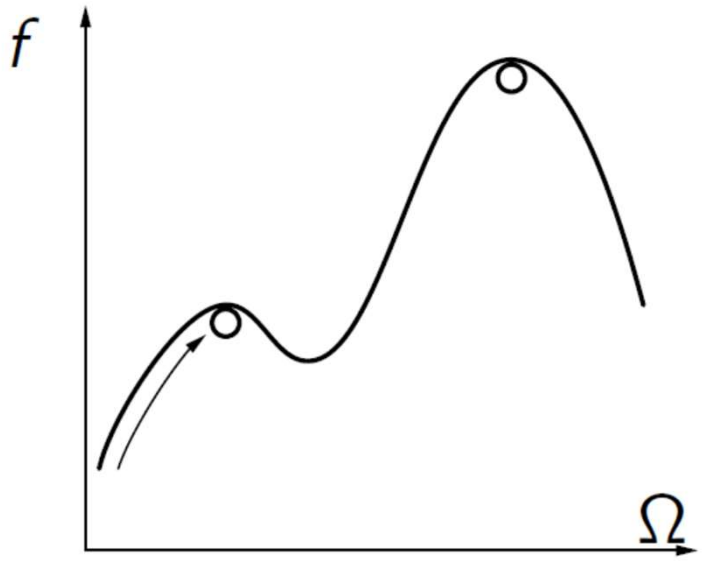
Output: **true** or **false**

1: **return** $B.F \succ A.F$

- Problem: Getting stuck in local maxima

Simulated Annealing (1)

- Extension of hill climbing and gradient ascent which avoids getting stuck
- Idea – Moving from lower into higher (local) maxima should be more probable than the opposite move



- Guiding principle
  - random variants of current solution are created
  - better solutions are always accepted
  - worse solutions are accepted with certain probability which depends on
    - quality difference between current and new solution
    - temperature parameter (decreases over the time)

Simulated Annealing (2)

1) Choose a (random) starting point $x_0 \in \Omega$
2) Choose a point $x' \in \Omega$ "in the vicinity" of x_t (e.g. by a small random variation of x_t)
3) Set

$$x_{t+1} = \begin{cases} x' & \text{if } f(x') \geq f(x_t) \\ x' \text{ with probability } p = e^{-\frac{\Delta f}{kT}} & \text{otherwise} \\ x_t \text{ with probability } 1 - p & \end{cases}$$

$\Delta f = f(x') - f(x_t)$	quality reduction of the solution
$k = \Delta f_{\max}$	estimate of the range of quality values
T	temperature parameter

4) Repeat steps 2 and 3 until a termination criterion is fulfilled

- For small T method is almost identical to hill climbing

Simulated Annealing (3)

Algorithm 3 Acceptance criterion of Simulated Annealing

Input: parental fitness $A.F$, fitness of offspring $B.F$, generation t

Output: **true** oder **false**

```
1: if B.F ≻ A.F {
2:     return true
3: } else {
4:     u ← choose randomly from U([0, 1])        /* random number
       between 0 and 1 */
5:     if u ≤ exp(−(A.F−B.F)/(kT_{t−1})) {
6:         return true
7:     } else {
8:         return false
9:     }
10: }
```

Threshold Accepting (1)

- Idea – Very similar to simulated annealing: worse solutions are sometimes accepted, but with an upper bound on quality degradation
  1) Choose a (random) starting point $x_0 \in \Omega$
  2) Choose a point $x' \in \Omega$ "in the vicinity" of x_t (e.g. by a small random variation of x_t)
  3) Set

$$x_{t+1} \begin{cases} x' & \text{if } f(x') \geq f(x_t) - \theta \\ x_t & \text{otherwise} \end{cases}$$

  θ threshold for accepting worse solution candidates

  θ is –slowly- decreased over time

  $\theta = 0$ is equivalent to standard hill climbing

  4) Repeat steps 2 and 3 until a termination criterion is fulfilled

Threshold Accepting (2)

Algorithm 4 Acceptance criterion of Threshold Accepting

Input: parental fitness $A.F$, fitness of offspring $B.F$, generation t

Output: **true** oder **false**

```
1: if B.F ≻ A.F oder A.F − B.F ≤ θ {
2:     return true
3: } else {
4:     return false
5: }
```

Great Deluge Algorithm (1)

- Idea – Very similar to simulated annealing: worse solutions are sometimes accepted, but absolute lower bound is used
  1) Choose a (random) starting point $x_0 \in \Omega$
  2) Choose a point $x' \in \Omega$ "in the vicinity" of x_t (e.g. by a small random variation of x_t)
  3) Set

$$x_{t+1} \begin{cases} x' & \text{if } f(x') \geq \theta_0 + t \cdot \eta \\ x_t & \text{otherwise} \end{cases}$$

  θ_0 lower bound for the quality of the candidate solutions at $t = 0$ (initial "water level"))

  η step width parameter ("speed of the rain")
  4) Repeat steps 2 and 3 until a termination criterion is fulfilled

Great Deluge Algorithm (2)

Algorithm 5 Acceptance criterion of Great Deluge Algorithm

Input: parental fitness $A.F$, fitness of offspring $B.F$, generation t

Output: **true** oder **false**

```
1: if B.F ≻ θ_0 + η · t {
2:     return true
3: } else {
4:     return false
5: }
```

Record-to-Record Travel (1)

- Idea – Similar to great deluge algorithm: rising water level is used, linked to the fitness of the best found individual
  - possibile degradation: always seen in relation to the best found individual
  - only if there is an improvement: current individual is important
  - similar to threshold accepting: a monotonously increasing sequence of real numbers controls the selection of poor individuals

Record-to-Record Travel (2)

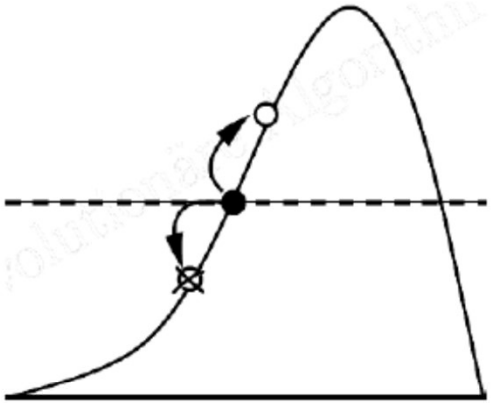
Algorithm 6 Acceptance criterion of Record-to-Record Travel

Input: parental fitness $A.F$, fitness of offspring $B.F$, t, best found quality F_{best}

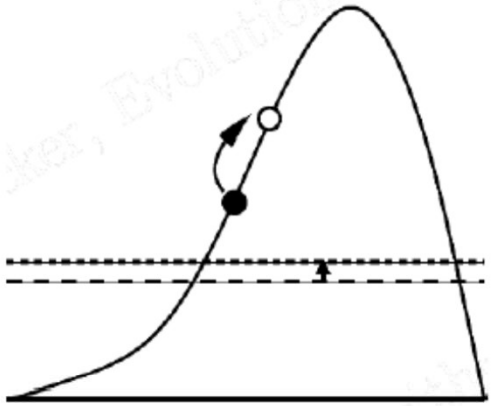
Output: **true** oder **false**, F_{best}

```
1: if B.F ≻ F_best {
2:     F_best ← B.F
3:     return true, F_best
4: } else {
5:     if B.F − F_best < T_t {
6:         return true, F_best
7:     }
8: }
9: return false, F_best
```

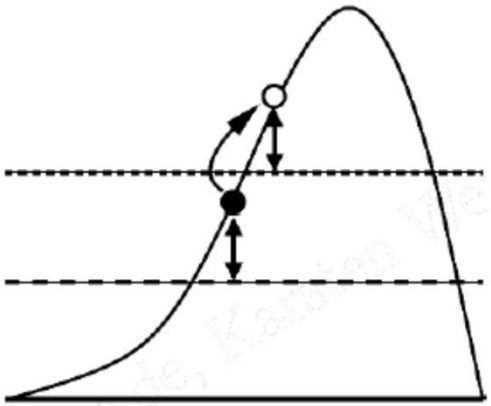
Comparison of local search algorithms



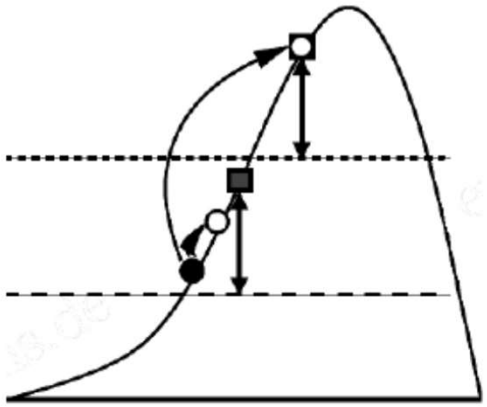
Hill Climbing



Great Deluge Algorithm



Threshold Accepting



Record-to-Record Travel

Example:
The Traveling Salesman Problem

Example: The Traveling Salesman Problem (1)

- Given: set of n cities (idealized as points on a plane) and distances/costs of the routes between the cities
- Desired: round trip with minimum distance through all n cities so that each city is visited exactly once
- Mathematically: find a so-called Hamiltonian cycle (contains each node once) with minimal total weight in a weighted graph
- Known: TSP is NP-complete ⇒ no algorithm is known that solves this problem in polynomial time
- Therefore: for large n only approximate solutions can be computed in reasonable time (best solution could be found in some cases, but not every time)
- Using hill climbing and simulated annealing to solve the problem

Example: The Traveling Salesman Problem (2)

1) Order the cities randomly (that is, create a random round trip)
2) Randomly choose two pairs of cities such that each pair consists of cities that are neighbors in the current round trip and such that all four cities are distinct, split the round trip between the cities of each pair and reverse the interjacent part
3) If this new round trip is better (that is, shorter or cheaper) than the old, then replace the old round trip with the new one.

   Otherwise replace the old round trip with probability $p = e^{-\frac{\Delta Q}{kT}}$

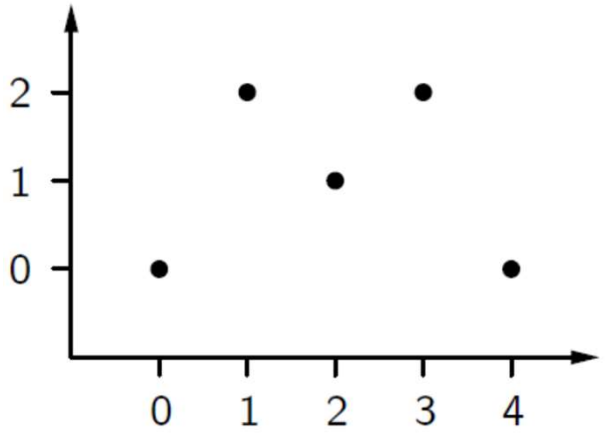
   ΔQ quality difference between the old and the new round trip

   k estimate of the range of round trip qualities, e.g. $k_t = \frac{t+1}{t} \max_{i=1}^{t} \Delta Q_i$, where ΔQ_i is the quality difference in the i-th step and t is the current step)

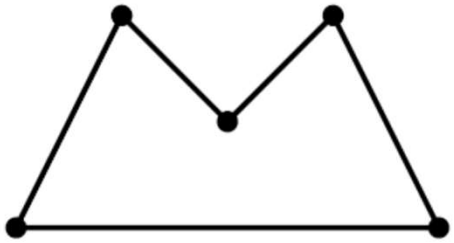
   T temperature parameter (reduced over time, e.g. $T = \frac{1}{t}$)
4) Repeat steps 2 and 3 until a termination criterion is met

Example: The Traveling Salesman Problem (3)

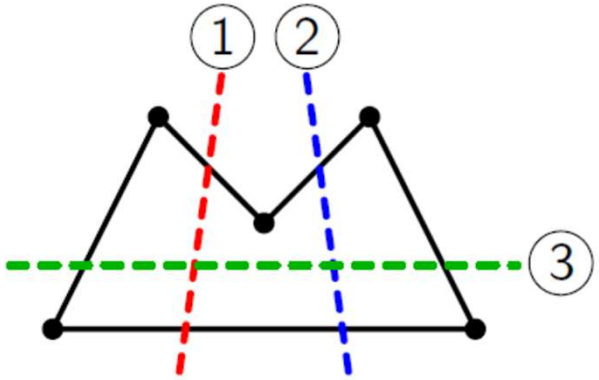
- Pure hill climbing can get stuck in local minimum



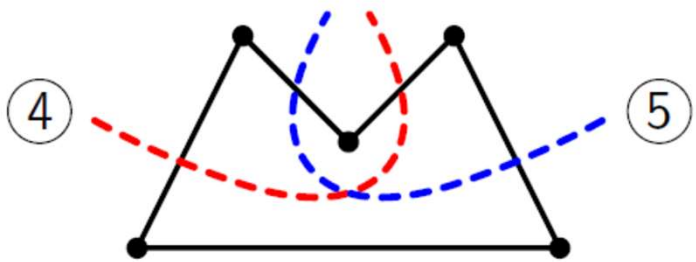
initial round trip



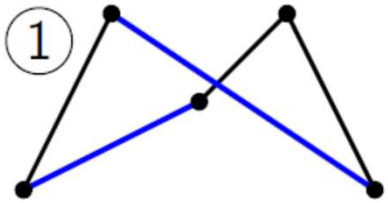
Length: $2\sqrt{2} + 2\sqrt{5} + 4 \approx 11.30$

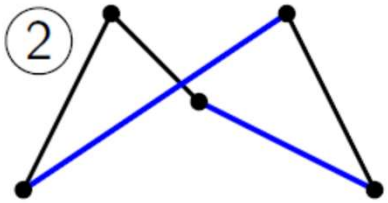


possible seperations of the round trip

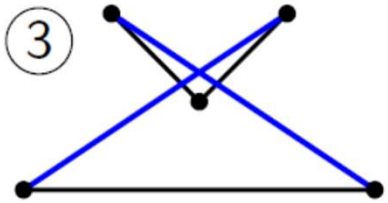


Example: The Traveling Salesman Problem (4)

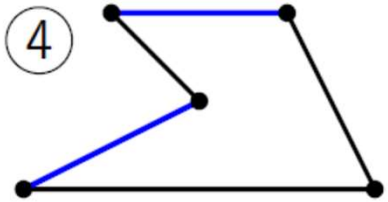


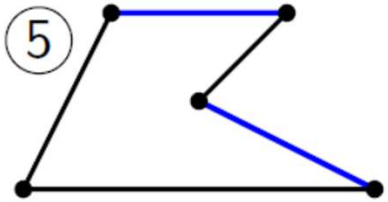


$\sqrt{2} + 3\sqrt{5} + \sqrt{13}$ ≈ 11.73



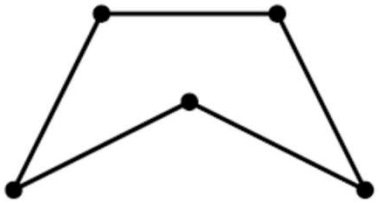
$\sqrt{2} + 2\sqrt{13} + 4$ ≈ 14.04





$\sqrt{2} + 2\sqrt{5} + 2 + 4$ ≈ 11.89

global optimum:



$4\sqrt{5} + 2$ ≈ 10.94

Example: The Traveling Salesman Problem (5)

- All modifications of the initial round trip results in worse round trips $\Longrightarrow$ global optimum (of this round trip) is not reachable by pure hill climbing
- Simulated annealing accepts sometimes worse solutions $\Longrightarrow$ way to find global optimum (but: no guarantee that will happen!)
- Be careful: it depends on permitted operations if search can get stuck in local optimum
  - When using another operation (change of the position of a city in round trip respect to delete of current position and insert on another one), getting stuck can be avoided in considered example
  - For this set of operations: construction of an example that gets stuck in local minimum

EA-related Methods

Tabu Search (1)

- Local search method that considers history when creating new individual
- Tabu lists avoid recurrence on previously considered solution candidates

- Tabu-list = FIFO queue with fixed length
  - Entries are complete solution candidate or aspects
  - Mutations are not allowed to create tabu entries
  - Often: FIFO list with beneficial properties can break tabu
  - Even possible: new best entire quality breaks tabu

Tabu Search (2)

Example

- Graph coloring with k colors so that two nodes connected with an edge are colored differently
  - $\Omega = \{1, \ldots, k\}^n$
  - Minimizing of $f(x) = \sum_{(v_i, v_j) \in E} \begin{cases} 1 & 1 \text{ if } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$
  - Mutating colors v_i from c to $d \Longrightarrow$ tabu entry (i, c)

Tabu Search (3)

Algorithm 7 Tabu search

Input: Target function F
Output: best individual A_{best}

1: $t \leftarrow 0$
2: $A(t) \leftarrow$ create random solution candidate
3: evaluate $A(t)$ by F
4: $A_{\text{best}} \leftarrow A(t)$
5: initalize Tabu-list
6: **while** termination criterion isn't fulfilled {
7: $P \leftarrow \emptyset$
8: **while** $|P| < \lambda$ {
9: $B \leftarrow$ mutate $A(t)$
10: evaluate B by F
11: **if** $(A(t), B) \notin$ Tabu-list or $B.F \succ A_{\text{best}}.F$ {
12: $P \leftarrow P \cup \{B\}$
13: }
14: }
15: $t \leftarrow t + 1$
16: $A(t) \leftarrow$ best individual of P
17: **if** $A(t).F \succ A_{\text{best}}.F$ {
18: $A_{\text{best}} \leftarrow A(t)$
19: }
20: Tabu-list $\leftarrow$ update by $(A(t-1), A(t))$
21: }
22: **return** A_{best}

Memetic Algorithms (1)

- Population-based algorithms
  - Advantage: widespread trawling of the search space
  - Disadvantage: slow
- Local search
  - Advantage: fast optimization
  - Disadvantage: susceptible to local optima
- Memetic Algorithms: Combination of both techniques
- Origin of the name: “memes” are elements of the behavior which can be acquired individually (in contrast to genes)
- Procedure: each new individual will be optimized immediately
  - Only few steps or
  - Till local optimum

Memetic Algorithms (2)

Algorithm 8 Memetic Algorithm

Input: Evaluation function F
Output: best individual

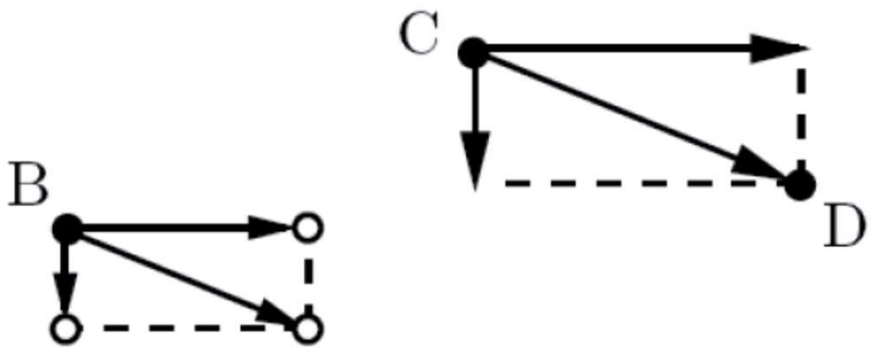
1: $t \leftarrow 0$
2: $P(t) \leftarrow$ initialize population of size μ
3: $P(t) \leftarrow$ LOCAL SEARCH(F) on each individual in $P(t)$
4: evaluate $P(t)$ by F
5: **while** termination criterion isn't fulfilled {
6: $E \leftarrow$ select parents for λ offspring in $P(t)$
7: $P' \leftarrow$ create offspring by recombination on E
8: $P'' \leftarrow$ mutate individual in P'
9: $P''' \leftarrow$ LOCAL SEARCH(F) on each individual in P''
10: evaluate P''' by F
11: $t \leftarrow t + 1$
12: $P(t) \leftarrow$ environmental selection on P'''
13: }
14: **return** best individual of $P(t)$

Memetic Algorithms (3)

- Often strongly accelerated optimization
- But: search dynamic can be limited crucially
  - Mutation gets stuck frequently in (wide) local optima
  - Recombination has bounded starting situation
  - Parts of the space are perhaps unreachable
- Method correlates with Lamarckian evolution

Differential Evolution (1)

- Idea
  - No adaptation of the step width in the set of strategies A.S
  - But: relations of individuals in the population can be used as basis for the step width



Differential Evolution (2)

Algorithm 9 DE-Operator

Input: individuals A, B, C, D

Output: optimized individual A'

1: index $\leftarrow$ choose random number according to $U(\{1, \ldots, l\})$
2: **for each** $i \in \{1, \ldots, l\}$ {
3: $u \leftarrow$ choose random number according to $U([0, 1))$
4: **if** $u \leq \tau$ **or** $i =$ index {
5: $A'.G_i \leftarrow B'.G_i + (C.G_i - D.G_i) \cdot \alpha$
6: } **else** {
7: $A'.G_i \leftarrow A.G_i$
8: }
9: }
10: **return** A'

Differential Evolution (3)

- DE-operator: combinations of recombination and mutation
- Selection: a new individual replaces parental individual if and only if it has a better fitness

Parameter	Range
Population size μ	10–100, $10 \cdot n$
Weighting of recombination τ	0.7–0.9
Scaling factor α	0.5–1.0

Differential Evolution (4)

Algorithm 10 Differentialevolution

Input: evaluation function F
Output: best individual of $P(t)$

1: $t \leftarrow 0$
2: $P(t) \leftarrow$ create population of size μ
3: evaluate $P(t)$ by F
4: **while** termination criterion isn't fulfilled {
5: $P(t+1) \leftarrow \emptyset$
6: **for** $i \leftarrow 1, \ldots, \mu$ {
7: **do** {
8: $A, B, C, D \leftarrow$ select parents uniformly random from $P(t)$
9: } **while** A, B, C, D pairwise distinct
10: $A' \leftarrow \text{DE-OPERATOR}(A, B, C, D)$
11: evaluate A' by F
12: **if** $F(A') \succ F(A)$ {
13: $P(t+1) \leftarrow P(t+1) \cup \{A'\}$
14: } **else** {
15: $P(t+1) \leftarrow P(t+1) \cup \{A\}$
16: }
17: }
18: }
19: $t \leftarrow t+1$
20: **return** best individual of $P(t)$

Scatter Search (1)

- Ingredients
  - Population with solution candidates
  - Variation operators
  - Selections pressure
  - Further: local search

- But...
  - A deterministic method
  - Broad exploration of the space
  - Wide initializing
  - Systematical generation of new individuals

Scattered Search (2)

- Iterative process with two phases
  1) Generating of new individuals and selection of those which guarantee the most possible variety
  2) Recombination of all "pairings" of the chosen individuals, selection of the best individuals and iteration until nothing changes anymore

Scattered Search (3)

STEP 1

1) Diversity generator creates μ individuals in P on the example
   - Per dimension of the space a value range containing four partitions
   - Notice of the frequency of the generated individuals per partition
   - Inverse proportionality in choosing the partition

2) Separate: population of the best α individuals is expanded by those β individuals of P which maximize $\min_{B \in P_{\text{best}}} d(A.G, B.G)\ (A \in P)$

Scattered Search (4)

STEP 2

1) Subset generator chooses individuals of the set with the best ones
2) Applying combination operator

   e.g.: arithmetic Crossover with $U\left(\left[-\frac{1}{2},\frac{3}{2}\right]\right)$
3) Local optimization
4) If all individuals are created $\Longrightarrow$ selection of the $\alpha+\beta$ best ones
5) Iterate until set is not changing anymore
6) α best individuals for next step 1

Scatter Search (5)

Algorithm 11 Scatter Search

Input: Evaluation function F

1: $P_{\text{best}} = \emptyset$; $P = \emptyset$
2: **for** $t \leftarrow 1, \ldots,$ maxIter {
3: **while** $|P| < \mu$ {
4: $A \leftarrow$ create individual with the diversity generator
5: $A \leftarrow$ LOCAL SEARCH(F) applied on A; evaluate A by F
6: **if** $A \notin P \cup P_{\text{best}}$ {
7: $P \leftarrow P \cup \{A\}$
8: }
9: }
10: **if** $t = 1$ {
11: $P_{\text{best}} \leftarrow$ select α individuals of P with BEST-SELECTION
12: $P \leftarrow$ discard individuals of P_{best} in P
13: }
14: **for** $k \leftarrow 1, \ldots, \beta$ {
15: $A \leftarrow$ the one individual of P that maximizes $\min_{B \in P_{\text{best}}} d(A.G, B.G)$
16: $P \leftarrow$ discard individual A in P
17: $P_{\text{best}} \leftarrow P_{\text{best}} \cup \{A\}$
18: }
19: **do** {
20: $P' \leftarrow \emptyset$; Sets $\leftarrow$ create subsets of P_{best} by using subset operator
21: **for each** $M \in$ Sets {
22: $A \leftarrow$ apply combination operator on M
23: $A \leftarrow$ LOCAL SEARCH(F) applied on A; evaluate A by F
24: **if** $A \notin P_{\text{best}} \cup P'$ {
25: $P' \leftarrow P' \cup \{A\}$
26: }
27: }
28: $P_{\text{best}} \leftarrow$ select $\alpha + \beta$ Ind. of $P_{\text{best}} \cup P'$ with BEST-SELECTION
29: } **while** P_{best} has not changed
30: $P_{\text{best}} \leftarrow$ select α individuals of P with BEST-SELECTION
31: }
32: **return** best individual in P_{best}

Scatter Search (6)

- Recommended parameters

Parameter value	Range
Population size μ	50–150
Total number of best individuals α	5–20
Expanding of best individuals β	5–20

Cultural Algorithm (1)

- Motivation:
  - Further information memory in addition to the genetic layer
  - Culture determines behavior based on certain values
  - Layer of culture is introduced in EAs

- Collectively cultural knowledge:
  - Memory: belief space
  - Is modified by the best individuals of a generation
  - Situational and normative knowledge

Cultural Algorithm (2)

Algorithm 12 Cultural Algorithm

Input: Evaluation function F

Output: best individual in $P(t)$

1: $t \leftarrow 0$
2: $P(t) \leftarrow$ initialize population
3: $\mathcal{BS}(t) \leftarrow$ initialize belief space
4: evaluate $P(t)$ by F
5: **while** termination criterion is not fulfilled $\{$
6: $\quad P' \leftarrow$ determine important individuals of $P(t)$
7: $\quad \mathcal{BS}(t+1) \leftarrow \mathcal{BS}(t)$ is adapted by P'
8: $\quad P'' \leftarrow$ create offspring of $P(t)$ on the basis of $\mathcal{BS}(t+1)$
9: $\quad$ evaluate $P(t)$ by F
10: $\quad t \leftarrow t+1$
11: $\quad P(t) \leftarrow$ selection from $P' \cup P(t-1)$
12: $\}$
13: **return** best individual of $P(t)$

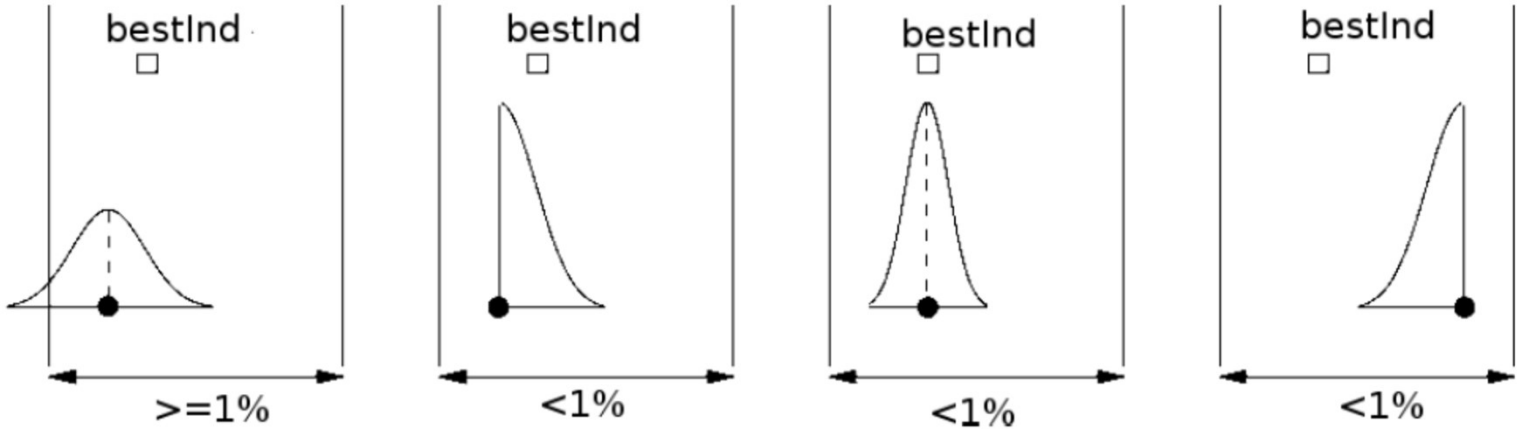
Cultural Algorithm (3)

- Situational knowledge
  - For $\Omega = \mathbb{R}^n$: two best individuals of the prior generation
  - Mutation should be orientated in those direction if necessary

- Normative knowledge
  - For $\Omega = \mathbb{R}^n$: upper and lower bound per dimension
  - Detect biggest/smallest value of the 20% best individuals of the prior generation
  - Always: accept increase
  - Only on "bound individual" with better quality: accept decrease

Cultural Algorithm (4)

- Stable space dimensions
  - If range is limited on < 1%
  - Contain last best individuals
- Mutation
  - If stable: orientate on best individual
  - Otherwise: self-adapted step width



Cultural Algorithm (5)

Algorithm 13 KA-Mutation

Input: Individual A
Output: Individual B

1: $u' \leftarrow$ choose randomly according to $\mathcal{N}(0,1)$
2: **for each** $i \in \{1,\ldots,l\}$ {
3: $\quad u_i'' \leftarrow$ choose randomly according to $\mathcal{N}(0,1)$
4: $\quad B.S_i \leftarrow A.S_i \cdot \exp\left(\frac{1}{\sqrt{2l}} \cdot u' + \frac{1}{\sqrt{2\sqrt{l}}} \cdot u_i''\right)$
5: $\quad u \leftarrow$ choose randomly according to $\mathcal{N}(0, B.S_i)$
6: $\quad$ **if** $\mathcal{BS}$ is stable on dimension i {
7: $\quad\quad$ **switch**
8: $\quad\quad$ **case** $A.G_i < \text{BestInd.G}_i : B.G_i \leftarrow A.G_i + |u|$
9: $\quad\quad$ **case** $A.G_i > \text{BestInd.G}_i : B.G_i \leftarrow A.G_i - |u|$
10: $\quad\quad$ **case** $A.G_i = \text{BestInd.G}_i : B.G_i \leftarrow A.G_i + \frac{u}{2}$
11: $\quad$ } **else** {
12: $\quad\quad B.G_i \leftarrow A.G_i + u$
13: $\quad\quad B.G_i \leftarrow \max\{\text{ug}_i, \min\{\text{og}_i, B.G_i\}\}$
14: $\quad$ }
15: }
16: **return** B