Introduction to Computational Intelligence

Winter 2010/11

Lecturer: Prof. Dr. Günter Rudolph

Stand-In: Nicola Beume

Computational Intelligence Group, LS11
Dept. of Computer Science
TU Dortmund

05.01.2011

Today's Topics

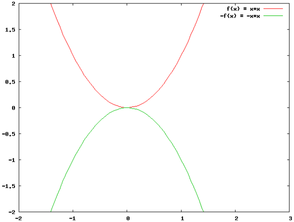
1. Optimization Basics
2. Randomized Search Heuristics
3. Introduction to Evolutionary Algorithms
    - EA Operators
4. Theory of Evolutionary Algorithms
    - Motivation
    - Method of Fitness-Based Partitions
    - Application of FBP
5. Summary and Outlook

Optimization Basics

given:
objective function $f : X \to \mathbb{R}$
feasible region X (= nonempty set)

objective: find solution with minimal or maximal value!

optimization problem:
find $\mathbf{x}^* \in X$ such that $f(\mathbf{x}^*) = \min\{f(\mathbf{x}) | \mathbf{x} \in X\}$
$\mathbf{x}^*$ global solution (optimizer)
$f(\mathbf{x}^*)$ global optimum (optimum)

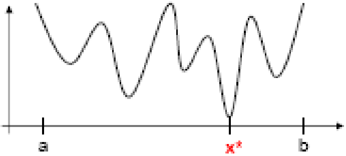


note: $\max\{f(x) | x \in X\} = -\min\{-f(x) | x \in X\}$

Optimization Basics

local optimum
$\mathbf{x}_l \in X$ is a local solution if
$\forall \mathbf{x} \in N(\mathbf{x}_l) : f(\mathbf{x}_l) \leq f(\mathbf{x})$
$N(\mathbf{x}_l)$ neighborhood of $\mathbf{x}_l$ (bounded subset of X)
$f(\mathbf{x}_l)$ local optimum, local minimum

note:
each global optimum is also a local one



“Easy” Classes of Optimization Problems

linear problems

- linear objective function, linear constraints
- solvable by e.g. simplex algorithms

non-linear problems

- objective function or constraints non-linear
- solvable by classical methods, if
- differentiable and
  - convex (convex function, convex domain)
  - without constraints
  - (more special cases...)

“Hard” Classes of Optimization Problems

What makes a problem hard

- local optima (is it a global optimum or not?)
- constraints (ill-shaped feasible region)
- non-smoothness (weak causality ⇒ strong causality needed!)
- discontinuities (⇒ nondifferentiability, no gradients)
- lack of knowledge about problem (⇒ black / gray box optimization)

Not solvable with conventional methods
⇒ use computational intelligence: randomized search heuristics

Classical algorithms vs. Randomized Search Heuristics

When to apply which method:

classical algorithms	rand. search heuristics
problem known: explicitly specified	problem unknown: given as black/gray box
problem well understood	problem poorly understood
problem-specific solver available	no problem-specific solver available
sufficient resources for designing algorithm affordable (time, experts)	insufficient human resources for designing algorithm, but oodles of computation time
solution with proven quality required	solution with satisfactory quality sufficient
⇝ **don't** apply RSH	⇝ try RSH

General Principles of Randomized Search Heuristics

View of Computer Science

optimization problems are search problems

- randomized
  decisions within algorithm performed probabilistically
- search
  optimal solution in space of feasible solutions
- heuristic
  strategy without proven quality
- black-box optimization
  algorithm doesn’t know the problem to optimize
  gets evaluation of quality for search points (externally)
  specific behavior depends on history of search points, evaluation

We consider evolutionary algorithms in the following...

Optimization in every day life

every day life problem:
fastest way from home to university?

try any way.
measure time.

change way slightly
try and measure time
in case of shorter time:
  remember way as favorite
repeat until satisfied

Optimization in every day life

every day life problem: :	optimization problem:
fastest way from home to university?	minimize travel time
try any way.	initialization
measure time.	function evaluation
	do:
change way slightly	generate variation
try and measure time	function evaluation
in case of shorter time:	
remember way as favorite	selection
repeat until satisfied	until stopping criterion fulfilled

this is an evolutionary algorithm!

Evolutionary Algorithms (EA)

inspired by biological evolution
considered as method of iterative improvements

Task
find $\mathbf{x} \in S$ optimizing some $f\colon S \to \mathbb{R}$.

- S search space
  feasible solution $\mathbf{x} \in S$
- f objective function used as fitness function, values/quality of solution

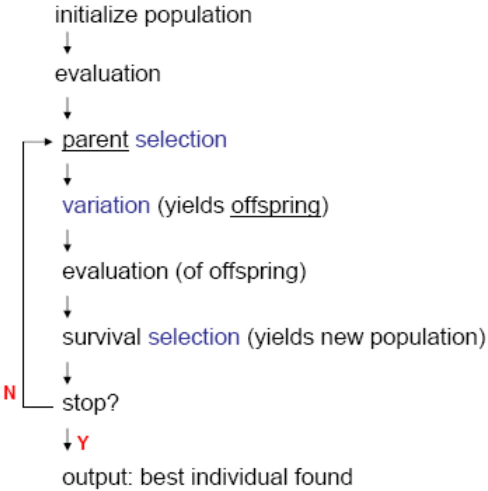
Often: $S = \mathbb{R}^n$ or $S = \mathbb{B}^n$ or $S = \mathbb{P}^n$ (permutations)

in this lecture today: $S = \mathbb{B}^n$

(Biological) Vocabulary

- genome (chromosome): search point, solution $\mathbf{x} = (x_1, \ldots, x_n)$
  decision variable, object parameter $x_i, i \in \{1, \ldots n\}$
  objective/fitness function value $y = f(\mathbf{x})$ of the optimization problem
- individual $\mathbf{a} = (\mathbf{x}, y)$: information bundle of solution
  population P_t: multiset of individuals in generation t
- genotype space: search space S of EA
  representation: encoding of genotype space ($\mathbb{R}^n, \mathbb{B}^n, \mathbb{P}^n$)
- reproduction: generation of search points by variation
- parent: individual used for reproduction
  offspring: new individual
- variation: recombination and/or mutation
  mutation: slight alteration of parent
  recombination/crossover: merging of several parents
- selection: choosing individuals
- generation: 1 iteration of EA

Algorithmic framework



initialize population

evaluation

parent selection

variation (yields offspring)

evaluation (of offspring)

survival selection (yields new population)

stop?

output: best individual found

Simple Example: (1+1)EA

```
$t = 0$                                          generation counter $t$
choose $\mathbf{x}_0 \in S$ uniformly at random  initialization
$y_0 = f(\mathbf{x}_0)$                          evaluation

Do                                               generation loop
   $\mathbf{x}'$ = mutation($\mathbf{x}_t$)         variation: mutation
   $y' = f(\mathbf{x}')$                            evaluation
   if $y' \leq y_t$                                 selection (minimization)
      $\mathbf{x}_{t+1} = \mathbf{x}'$; $y_{t+1} = y'$
   otherwise                                        subsequent population: solutions $\mathbf{x}_{t+1}$
      $\mathbf{x}_{t+1} = \mathbf{x}_t$; $y_{t+1} = y_t$
   $t = t + 1$                                      increase generation counter
stopping criterion fulfilled                     stopping criterion
```

Selection

population $P = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\mu)$ with μ individuals

selection at 2 steps of EA
  selection for reproduction: choose parents
  selection for survival: choose individuals for subsequent population

two approaches

1. repeatedly select individuals from population with replacement
2. rank individuals somehow and choose those with best ranks (no replacement)

uniform selection
  choose individual uniformly at random

truncation selection (deterministic)
  rank individuals according to fitness
  choose best individuals
  plus-selection: choose from current population and offspring, $(\mu + \lambda)$
  comma-selection: choose from offspring only, (μ, λ)

Mutation in search space $\mathbb{B}^n$

first: copy parent $\mathbf{x}$ to $\mathbf{x}'$

standard bit mutation

invert (flip) each bit x_i' independently with probability p_m

- expected number of inverted bits $= p_m \cdot n$
- $p_m \in (0; 1/2]$ to favor small changes
- most often used mutation probability $p_m = 1/n$

k-bit mutation

choose randomly uniformly k different positions in $\mathbf{x}'$, and invert these bits

- k often very small, most often $k = 1$
- easier to analyze than standard-bit-mutation
- behavior can vary greatly from standard-bit-mutation

Recombination/ Crossover in search space $\mathbb{B}^n$

discrete recombination

copy values (unchanged) from parents

k-point-crossover

choose 2 parents, choose k different positions uniformly at random
copy parts from parents alternatingly
most often k very small, usually $k = 2$ or $k = 1$

uniform crossover

choose ρ parents,
for every $\mathbf{x}'_i$: choose uniformly at random among parents
  which parent value $\mathbf{x}_i^{(j)}$, $j \in \{1, \ldots, \rho\}$ to copy
number of parent usually $\rho = 2$

Theory of Evolutionary Algorithms

What do we do if we design a problem-specific algorithm?

- prove its correctness (problem solved to optimality)
- analyze its performance: (expected) run time

What does this mean for optimization with evolutionary algorithms?

- prove that best function value in population converges to global optimum of problem f for generations $t \to \infty$
- analyze how long this takes on average: expected optimization time
- runtime measure: number of function evaluations
  black-box evaluation can afford huge resources (execute simulator, build machine, ...)
  making all other algorithmic steps of the EA marginal

Analysis of Evolutionary Algorithms

What kind of evolutionary algorithms do we want to analyze?

clearly	all kinds of evolutionary algorithms
more realistic	very simple evolutionary algorithms at least as starting point

For what kind of problems do we want to do analyses?

clearly	all kinds of problems
more realistic	very simple problems — “toy problems” at least as starting point

On “Toy Problems”

better term example problems

Why should we care?

- support analysis, help to develop analytical tools
- are easy to understand, are clearly structured
- present typical situations in a paradigmatic way
- make important aspects visible
- act as counter examples
- help to discover general properties
- are important tools for further design and analysis

Simple Scenario

EA: (1+1)EA
search space: $\mathbb{B}^n$

properties

- Hamming distance of 2 vectors: # of differing bits
  $H(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} (x_i + x'_i - 2x_i x'_i)$
- standard bit mutation with $p_m = 1/n$
  typical probabilities:
    Pr(specific bit flips) $= 1/n$
    Pr(specific bit doesn't flip) $= 1 - 1/n$
  E(#mutating bits)$= n \cdot 1/n = 1$
- plus-selection elitistic: no worsenings

example function: $\textsc{OneMax}(\mathbf{x}) = \sum_{i=1}^{n} x_i$

properties

- maximization, optimum: $\textsc{OneMax}(1^n) = n$
- 1 global optimum (no other local ones)

Fundamental Basics of Calculation with Probabilities

analyses by "puzzling" of good/bad basic events

event occurs with probability p
$\Rightarrow$ counter event has probability $1 - p$

connection of events by "OR" $\Rightarrow$ add probabilities

connection of events by "AND" $\Rightarrow$ multiply probabilities

lower bound of probability: leave out probability of some "OR"-events

upper bound of probability: leave out probability of some "AND"-events

here: discrete probability space $\Rightarrow$ combinatoric

number of combinations without order: binomial coefficient $\binom{n}{k}$
used in the following to count how many vector configurations
fulfill a certain condition
example: # of possible vectors of length 10 with exactly 3 0-bits: $\binom{10}{3}$

Upper Bounds with Fitness-Based Partitions (FBP)

method of fitness-based partitions works well with plus-selection for upper bounds on runtime

- group search points with equal/similar fitness in partition
- rank partitions according to ascending fitness values
- all elements of highest partition optimal
- selection elitistic: leave partition only towards better one
- worst case perspective to gain upper bound: initialize in worst partition
- sum up time spend in each partition until highest reached

Definition

Let $f\colon \{0,1\}^n \to \mathbb{R}$. A partition $L_0, L_1, \ldots, L_k$ of $\{0,1\}^n$ is called f-based partition iff the following holds.

1. $\forall i, j \in \{0, \ldots, k\} \colon \forall x \in L_i \colon \forall y \in L_j \colon \; (i < j \Rightarrow f(x) < f(y))$
2. $L_k = \{x \in \{0,1\}^n \mid f(x) = \max\{f(y) \mid y \in \{0,1\}^n\}\}$

Upper Bounds with Fitness-Based Partitions (FBP)

$Pr(\mathbf{x}$ mutates to $\mathbf{x}')$: $p_m^{H(\mathbf{x},\mathbf{x}')} \cdot (1-p_m)^{n-H(\mathbf{x},\mathbf{x}')}$
mutate $H(\mathbf{x},\mathbf{x}')$ bits, do not mutate $n - H(\mathbf{x},\mathbf{x}')$ bits

s_i : probability of leaving partition L_i
$s_i = \min_{\mathbf{x} \in L_i} \sum_{i<j\leq k} \sum_{\mathbf{x}' \in L_j} p_m^{H(\mathbf{x},\mathbf{x}')} \cdot (1-p_m)^{n-H(\mathbf{x},\mathbf{x}')}$
inner sum: all $\mathbf{x}'$ of higher partition L_j
outer sum: all higher partitions
min: worst $\mathbf{x}$

expected optimization time: sum of duration per partition
duration = 1/ (probability of leaving) $= s_i^{-1}$
lower bound of s_i leads to upper bound of s_i^{-1}

$$E(T_{(1+1)EA,f}) \leq \sum_{0\leq i<k} s_i^{-1}$$

Upper Bound for (1+1)EA on ONEMAX

use trivial partition: 1 partition for each function value acc. to ONEMAX

useful inequality: $(1-1/n)^n < 1/e < (1-1/n)^{n-1}$, e: Euler's number

vectors in partition L_i: i 1-bits, $n-i$ 0-bits
possible improvement: mutate one 0 → 1, other bits unchanged
⇒ function increased by 1 ⇒ partition left

Pr$(0 \to 1) =$ #0-bits $\cdot p_m = \binom{n-i}{1} \cdot 1/n = (n-i)/n$
Pr(other bits do not mutate)= $(1-p_m)^{n-1} = (1-1/n)^{n-1} > 1/e$

lower bound for probability of leaving partition:
$s_i \geq \frac{n-i}{n} \cdot (1-\frac{1}{n})^{n-1} \geq \frac{n-i}{n} \cdot \frac{1}{e} = \frac{n-i}{ne}$

$$E(T_{(1+1)EA,\text{ONEMAX}}) \leq \sum_{0\leq i<n} s_i^{-1} \leq \sum_{0\leq i<n} \frac{en}{n-i} = en \sum_{1\leq i\leq n} \frac{1}{i}$$
$$= enH_n < en(\ln(n)+1) = O(n\log n)$$

Upper Bound: (1+1) EA on LEADINGONES

LEADINGONES: $\{0,1\}^n \to \mathbb{R}$ with LEADINGONES$(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$

use trivial partition: 1 partition for each function value acc. to LEADINGONES

improving step:
to leave L_i by one mutation, flip exactly the leftmost 0-bit.

$s_i \geq 1 \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en}$

$$\begin{aligned} \text{E}\left(T_{\text{(1+1) EA,LEADINGONES}}\right) &\leq \sum_{i=0}^{n-1} s_i^{-1} = \sum_{i=0}^{n-1} en = n \cdot en \\ &= O(n^2) \end{aligned}$$

Summary and Outlook

Summary

- randomized search heuristics suitable tool for complex problems
- evolutionary algorithms (EA): basic operators
- simple example: (1+1)-EA
- theory possible

Upcoming topics, e.g.

- evolutionary algorithms with search space $\mathbb{R}^n$
- design principles of EA
- parameters

Acknowledgments:
lecture based on slides by Günter Rudolph, Thomas Jansen

Thanks!