Introduction to Optimization

Lecture 3: Approximation Algorithms and Heuristics

October 2, 2015

TC2 - Optimisation

Université Paris-Saclay, Orsay, France

Anne Auger
INRIA Saclay – Ile-de-France

Dimo Brockhoff
INRIA Lille – Nord Europe

Course Overview

Date		Topic
Fri, 18.9.2015	DB	Introduction and Greedy Algorithms
Fri, 25.9.2015	DB	Dynamic programming and Branch and Bound
Fri, 2.10.2015	DB	Approximation Algorithms and Heuristics
Fri, 9.10.2015	AA	Introduction to Continuous Optimization
Fri, 16.10.2015	AA	End of Intro to Cont. Opt. + Gradient-Based Algorithms I
Fri, 30.10.2015	AA	Gradient-Based Algorithms II
Fri, 6.11.2015	AA	Stochastic Algorithms and Derivative-free Optimization
16 - 20.11.2015		Exam (exact date to be confirmed)

all classes + exam are from 14h till 17h15 (incl. a 15min break)
here in PUIO-D101/D103

Overview of Today's Lecture

Approximation Algorithms

- a greedy approximation algorithm for bin packing
- an FPTAS for the KP

Overview of (Randomized) Search Heuristics

- randomized local search
- variable neighborhood search
- tabu search
- evolutionary algorithms
- exercise: an evolutionary algorithm for the knapsack problem

Approximation Algorithms

Coping with Difficult Problems

Exact

- brute-force often too slow
- better strategies such as dynamic programming & branch and bound
- still: often exponential runtime

Approximation Algorithms (now)

- guarantee of low run time
- guarantee of high quality solution
- obstacle: difficult to prove these guarantees

Heuristics (later today)

- intuitive algorithms
- guarantee to run in short time
- often no guarantees on solution quality

Approximations, PTAS, and FPTAS

- An algorithm is a *ρ-approximation algorithm* for problem Π if, for each problem instance of Π, it outputs a feasible solution which function value is within a ratio ρ of the true optimum for that instance.

- An algorithm A is an *approximation scheme* for a *minimization** problem Π if for any instance I of Π and a parameter $\varepsilon>0$, it outputs a solution s with $f_{\Pi}(I,s) \leq (1+\varepsilon) \cdot OPT$.

- An approximation scheme is called *polynomial time approximation scheme (PTAS)* if for a fixed $\varepsilon>0$, its running time is polynomially bounded in the size of the instance I.
  - note: runtime might be exponential in $1/\varepsilon$ actually!

- An approximation scheme is a *fully polynomial time approximation scheme (FPTAS)* if its runtime is bounded polynomially in both the size of the input I and in $1/\varepsilon$.

Today's Lecture

- only example algorithm(s)
- no detailed proofs here due to the time restrictions of the class
- rather spend more time on the heuristics part and the exercise

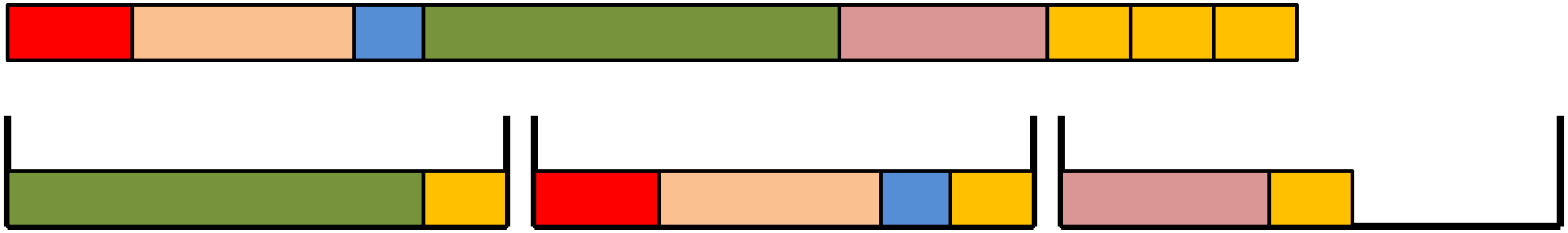
Actually Two Examples:

- a greedy approximation algorithm for bin packing
- an FPTAS for the KP

Bin Packing (BP)

Bin Packing Problem

Given a set of n items with sizes a_1, a_2, ..., a_n. Find an assignment of the a_i's to bins of size V such that the number of bins is minimal and the sum of the sizes of all items assigned to each bin is $\leq$ V.



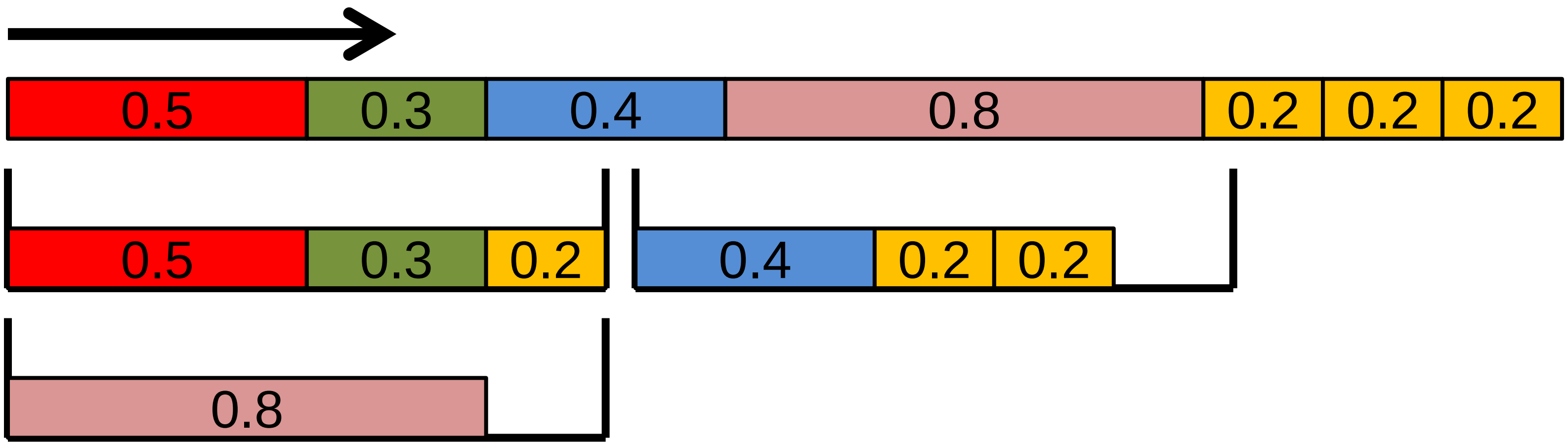
Known Facts

- no optimization algorithm reaches a better than 3/2 approximation in polynomial time (not shown here)
- greedy first-fit approach already yields an approximation algorithm with ρ-ratio of 2

First-Fit Approach

First-Fit Algorithm

- without sorting the items do:
  - put each item into the first bin where it fits
  - if it does not fit anywhere, open a new bin



Theorem: First-Fit algorithm is a 2-approximation algorithm

Proof: Assume First Fit uses m bins. Then, at least m-1 bins are more than half full (otherwise, move items).

$$\mathrm{OPT} > \frac{m-1}{2} \iff 2\mathrm{OPT} > m-1 \implies 2\mathrm{OPT} \geq m$$

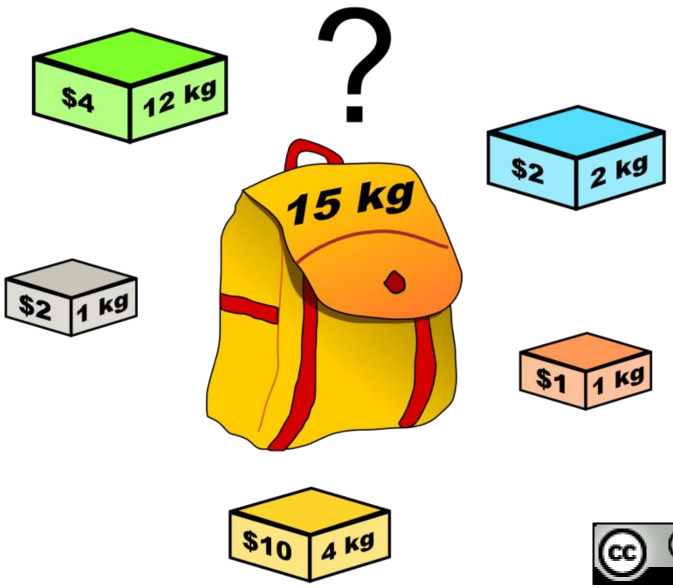
↑because m and OPT are integer

An FPTAS for the Knapsack Problem

Knapsack Problem

$$\max. \sum_{j=1}^{n} p_j x_j \text{ with } x_j \in \{0,1\}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq W$$



An FPTAS for the Knapsack Problem

Similar to last week's exercise, we can design a dynamic programming algorithm for which

- a subproblem is restricting the items to {1, ..., k} and searches for the lightest packing with prefixed profit P
- runs in $O(n^2 P_{max})$

What strange runtime is $O(n^2 P_{max})$?

Answer: pseudo-polynomial (polynomial if P_{max} would be polynomial in input size)

Idea behind FPTAS:

- scale the profits smartly to $\left\lfloor \frac{p_i n}{\varepsilon P_{max}} \right\rfloor$ to make P_{max} polynomially bounded
- prove that dynamic programming approach computes profit of at least $(1-\varepsilon)\cdot OPT$ (not shown here)

(Randomized) Search Heuristics

Motivation General Search Heuristics

- often, problem complicated and not much time available to develop a problem-specific algorithm
- search heuristics are a good choice:
  - relatively easy to implement
  - easy to adapt/change/improve
    - e.g. when the problem formulation changes in an early product design phase
    - or when slightly different problems need to be solved over time
- randomized/stochastic algorithms are a good choice because they are robust to noise

Lecture Overview

- Randomized Local Search (RLS)
- Variable Neighborhood Search (VNS)
- Tabu Search (TS)
- Evolutionary Algorithms (EAs)

Neighborhoods

For most (stochastic) search heuristics, we need to define a *neighborhood structure*

- which search points are close to each other?

Example: k-bit flip / Hamming distance k neighborhood

- search space: bitstrings of length n ($\Omega=\{0,1\}^n$)
- two search points are neighbors if their Hamming distance is k
- in other words: x and y are neighbors if we can flip exactly k bits in x to obtain y
- 0001001101 is neighbor of
  0001000101 for k=1
  0101000101 for k=2
  1101000101 for k=3

Randomized Local Search (RLS)

Idea behind (Randomized) Local Search:

- explore the local neighborhood of the current solution (randomly)

Pure Random Search:

- go to randomly chosen neighbor

First Improvement Local Search:

- go to first (randomly) chosen neighbor which is better

Best Improvement strategy:

- always go to the best neighbor
- not random anymore
- computationally expensive if neighborhood large

Variable Neighborhood Search

Main Idea: [Mladenovic and P. Hansen, 1997]

- change the neighborhood from time to time
  - local optima are not the same for different neighborhood operators
  - but often close to each other
  - global optimum is local optimum for all neighborhoods
- rather a framework than a concrete algorithm
  - e.g. deterministic and stochastic neighborhood changes
- typically combined with (i) first improvement, (ii) a random order in which the neighbors are visited and (iii) restarts

N. Mladenovic and P. Hansen (1997). "Variable neighborhood search". Computers and Operations Research 24 (11): 1097–1100.

Tabu Search

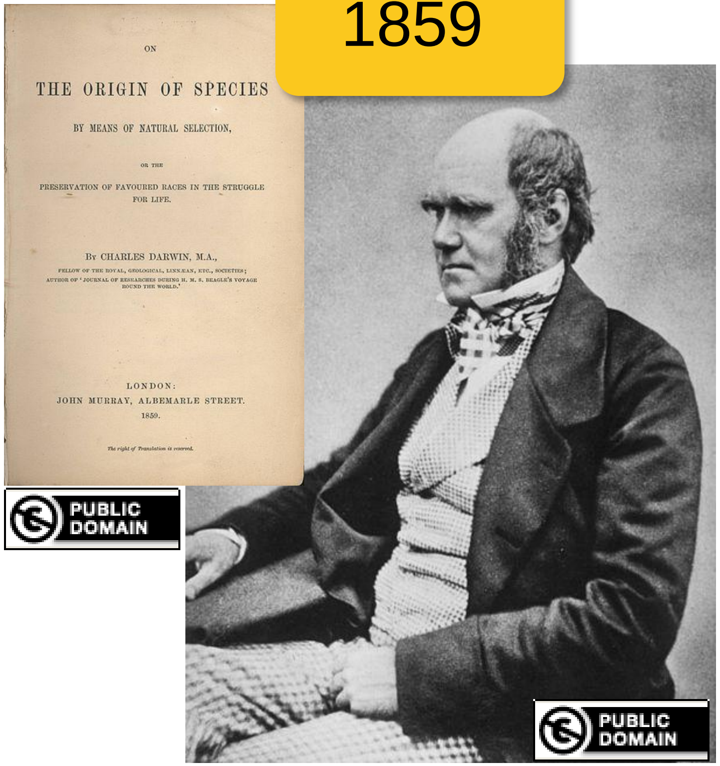
Disadvantages of local searches (with or without varying neighborhoods)

- they get stuck in local optima
- have problems to traverse large plateaus of equal objective function value (“random walk”)

Tabu search addresses these by

- allowing worsening moves if all neighbors are explored
- introducing a tabu list of temporarily not allowed moves
- those restricted moves are
  - problem-specific and
  - can be specific solutions or not permitted “search directions” such as “don’t include this edge anymore” or “do not flip this specific bit”
- the tabu list is typically restricted in size and after a while, restricted moves are permitted again

One class of (bio-inspired) stochastic optimization algorithms: Evolutionary Algorithms (EAs)



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

By CHARLES DARWIN, M.A.,

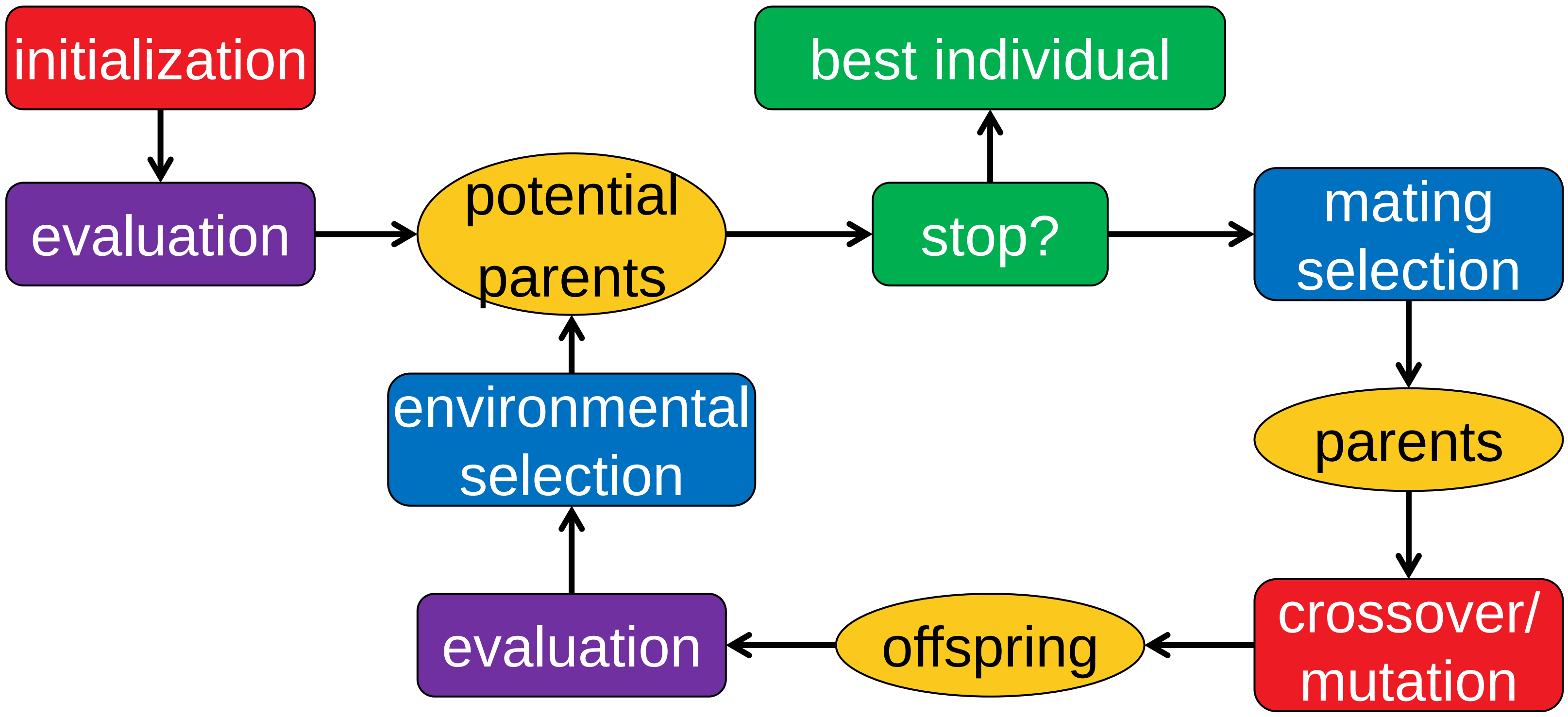
LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

- Class of optimization algorithms originally inspired by the idea of biological evolution
- selection, mutation, recombination

Metaphors

Classical Optimization	Evolutionary Computation
variables or parameters	variables or chromosomes
candidate solution vector of decision variables / design variables / object variables	individual, offspring, parent
set of candidate solutions	population
objective function loss function cost function error function	fitness function
iteration	generation

Generic Framework of an EA



stochastic operators

"Darwinism"

stopping criteria

Important:
representation (search space)

Genetic Algorithms (GA)

J. Holland 1975 and D. Goldberg (USA)

$$\Omega = \{0, 1\}^n$$

Evolution Strategies (ES)

I. Rechenberg and H.P. Schwefel, 1965 (Berlin)

$$\Omega = \mathbb{R}^n$$

Evolutionary Programming (EP)

L.J. Fogel 1966 (USA)

Genetic Programming (GP)

J. Koza 1990 (USA)

$$\Omega = \text{space of all programs}$$

nowadays one umbrella term: evolutionary algorithms

Genotype – Phenotype mapping

The genotype – phenotype mapping

- related to the question: how to come up with a fitness of each individual from the representation?
- related to DNA vs. actual animal (which then has a fitness)

fitness of an individual not always = f(x)

- include constraints
- include diversity
- others
- but needed: always a total order on the solutions

Handling Constraints

Several possible ways to handle constraints, e.g.:

- resampling until a new feasible point is found ("often bad idea")
- penalty function approach: add constraint violation term (potentially scaled)
- repair approach: after generation of a new point, repair it (e.g. with a heuristic) to become feasible again if infeasible
  - continue to use repaired solution in the population or
  - use repaired solution only for the evaluation?
- multiobjective approach: keep objective function and constraint functions separate and try to optimize all of them in parallel
- many more...

Examples for some EA parts

Selection

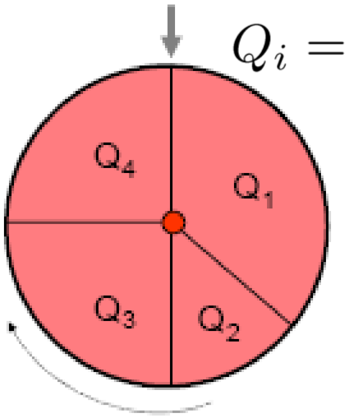
Selection is the major determinant for specifying the trade-off between exploitation and exploration

Selection is either

stochastic or deterministic

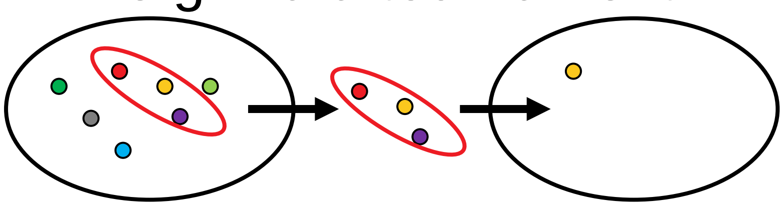
e.g. fitness proportional

$$Q_i = \frac{f(x_i)}{\sum_{j=1}^{\mu} f(x_j)}$$



Disadvantage: depends on scaling of f

e.g. via a tournament



e.g. (μ+λ), (μ,λ)

best μ from offspring *and* parents

best μ from offspring only

Mating selection (selection for variation): usually stochastic

Environmental selection (selection for survival): often deterministic

Variation Operators

Variation aims at generating new individuals on the basis of those individuals selected for mating

Variation = Mutation and Recombination/Crossover

mutation: *mut:* $\Omega \to \Omega$

recombination: *recomb:* $\Omega^r \to \Omega^s$ where $r \geq 2$ and $s \geq 1$

- choice always depends on the problem and the chosen representation
- however, there are some operators that are applicable to a wide range of problems and tailored to standard representations such as vectors, permutations, trees, etc.

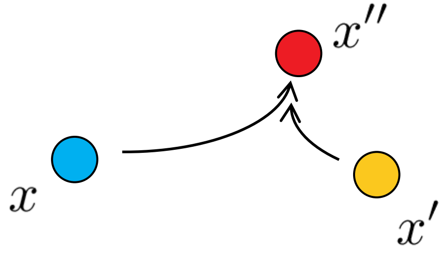
Variation Operators: Guidelines

Two desirable properties for mutation operators:

- every solution can be generation from every other with a probability greater than 0 (“exhaustiveness”)
- $d(x, x') < d(x, x'') => Prob(\mathrm{mut}(x) = x') > Prob(\mathrm{mut}(x) = x'')$ (“locality”)

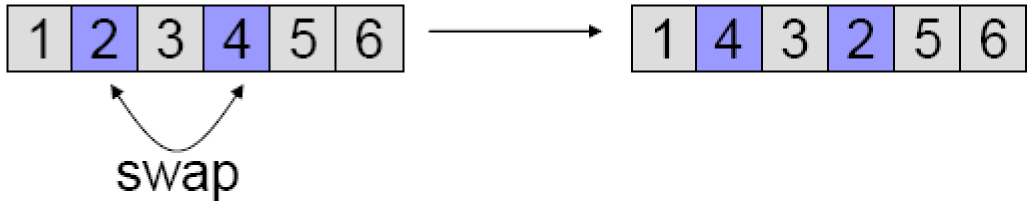
Desirable property of recombination operators (“in-between-ness”):

$$x'' = \ \mathrm{recomb}(x, x') \Rightarrow d(x'', x) \leq d(x, x') \wedge d(x'', x') \leq d(x, x')$$

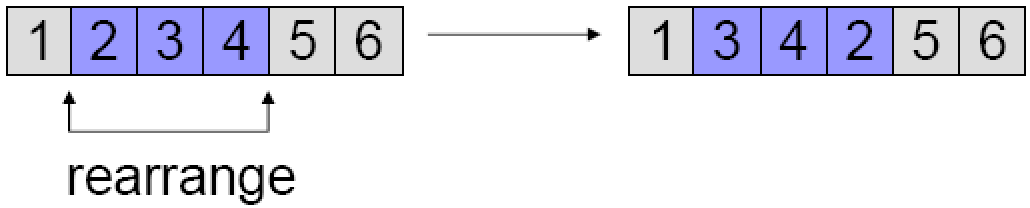


Examples of Mutation Operators on Permutations

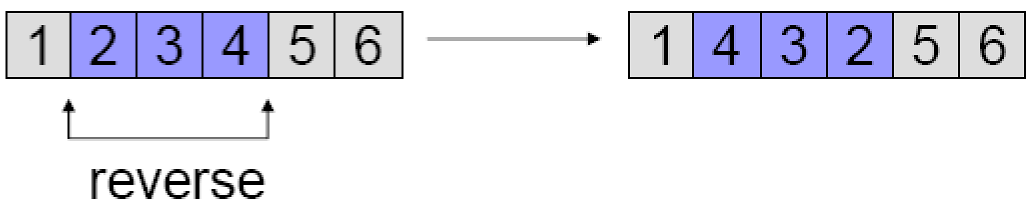
Swap:



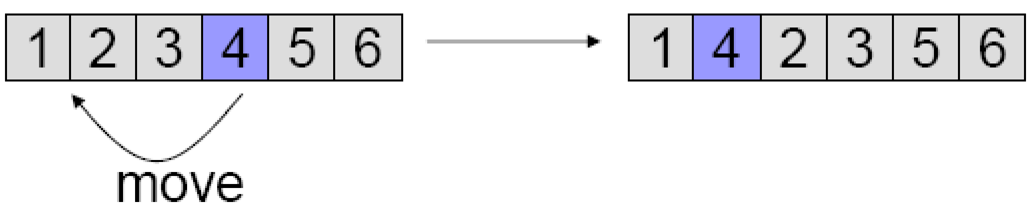
Scramble:



Invert:

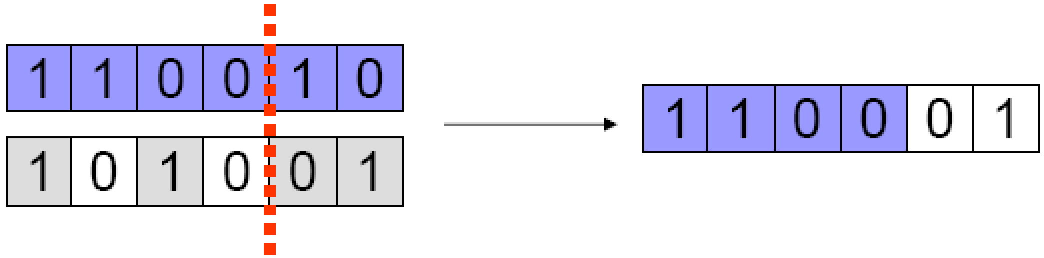


Insert:

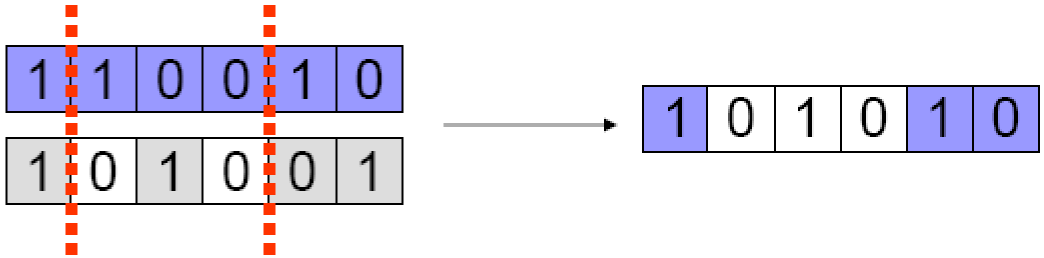


Examples of Recombination Operators: $\{0,1\}^n$

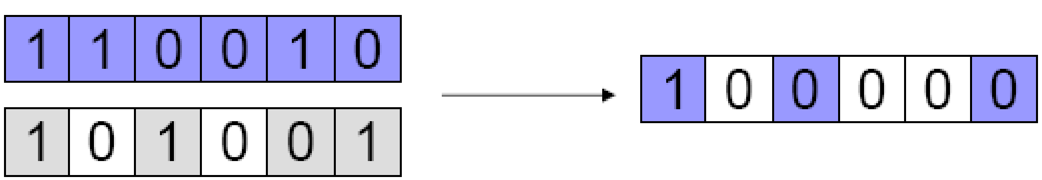
1-point crossover



n-point crossover



uniform crossover



choose each bit independently from one parent or another

A Canonical Genetic Algorithm

- binary search space, maximization
- uniform initialization
- generational cycle: of the population
  - evaluation of solutions
  - mating selection (e.g. roulette wheel)
  - crossover (e.g. 1-point)
  - environmental selection (e.g. plus-selection)

Conclusions

- EAs are generic algorithms (randomized search heuristics, meta-heuristics, ...) for black box optimization

  no or almost no assumptions on the objective function

- They are typically less efficient than problem-specific (exact) algorithms (in terms of #funevals)

  not the case in the continuous case (we will see later)

- Allow for an easy and rapid implementation and therefore to find good solutions fast

  easy to incorporate (and recommended!) to incorporate problem-specific knowledge to improve the algorithm

Exercise:
An Evolutionary Algorithm for the Knapsack Problem

`http://researchers.lille.inria.fr/~brockhof/optimizationSaclay/`

Conclusions

I hope it became clear...

...that approximation algorithms are often what we can hope for in practice (might be difficult to achieve guarantees though)

...that heuristics is what we typically can afford in practice (no guarantees and no proofs)

...and how to apply the evolutionary algorithm paradigm to the knapsack problem