Machine Learning and Pervasive Computing

Stephan Sigg

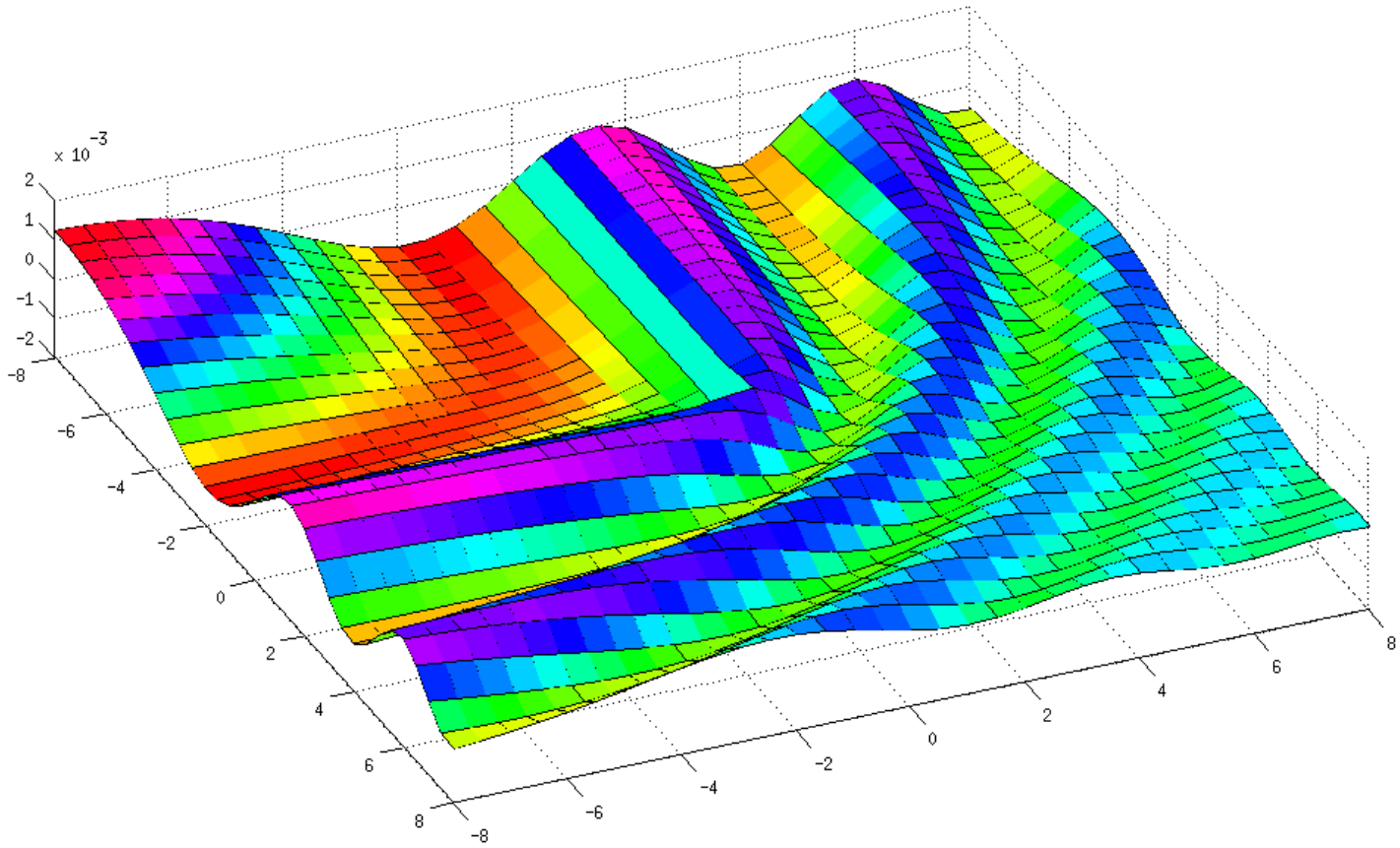
Georg-August-University Goettingen, Computer Networks

13.05.2015

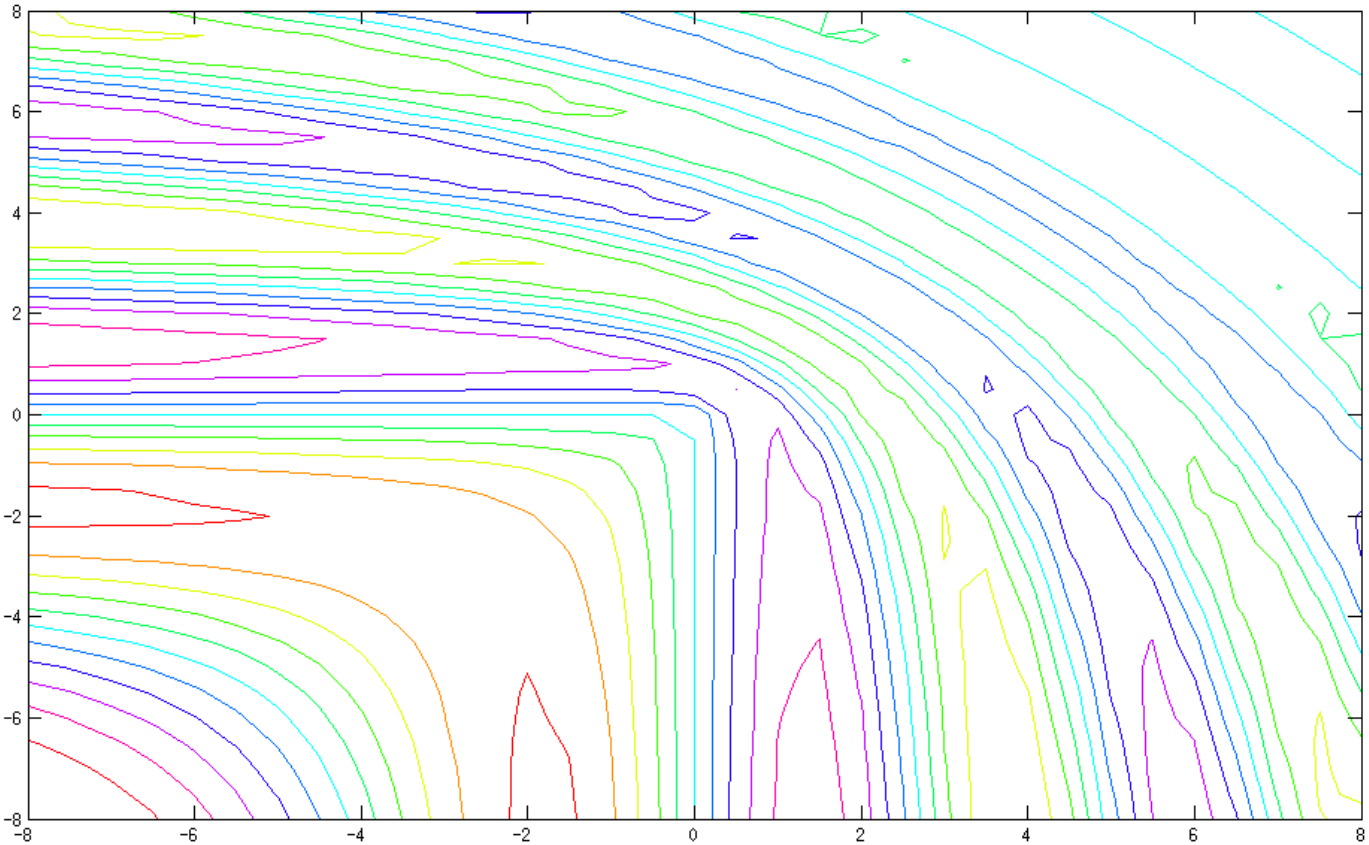
Overview and Structure

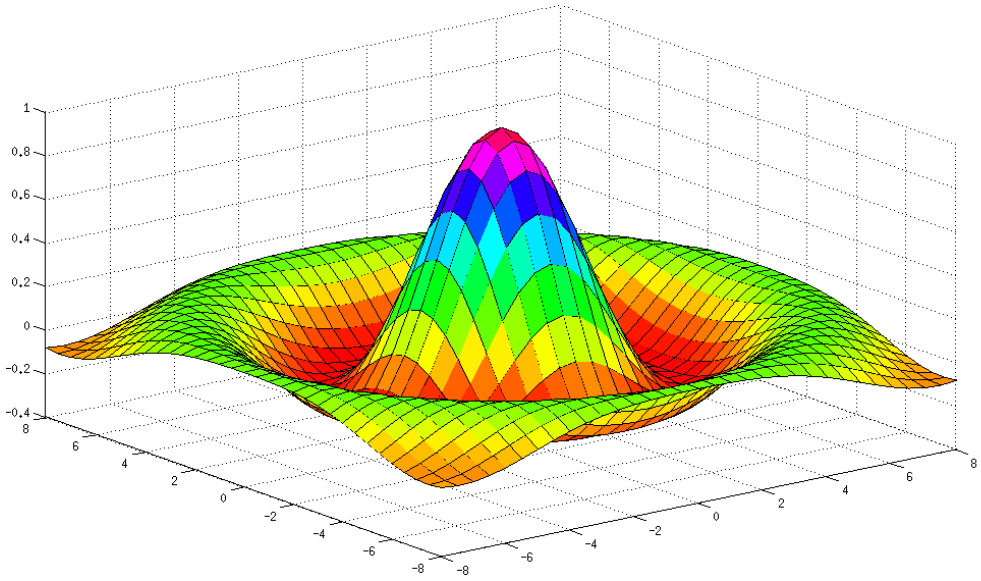
15.04.2015	Organisation
15.04.2015	Introduction
22.04.2015	–
29.04.2015	Rule-based learning and Decision Trees
06.05.2015	A simple Supervised learning algorithm
13.05.2015	Excursion: Avoiding local optima with random search
20.05.2015	High dimensional data
27.05.2015	–
03.06.2015	Artificial Neural Networks
10.06.2015	–
17.06.2015	k-Nearest Neighbour methods
24.06.2015	Probabilistic models
01.07.2015	Topic models
08.07.2015	Unsupervised learning
15.07.2015	Anomaly detection, Online learning, Recom. systems
22.07.2015	Q+A

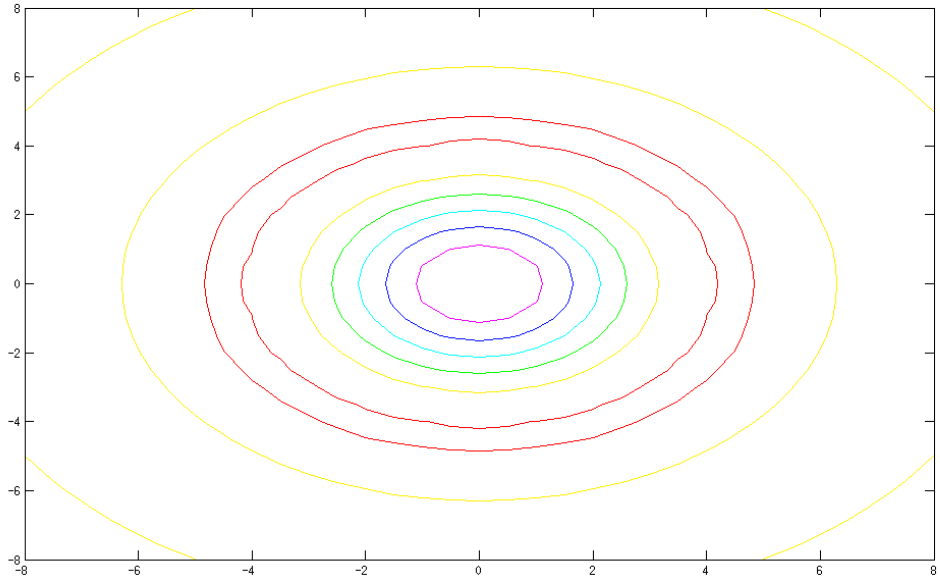
Local optima



Local optima – contour plot

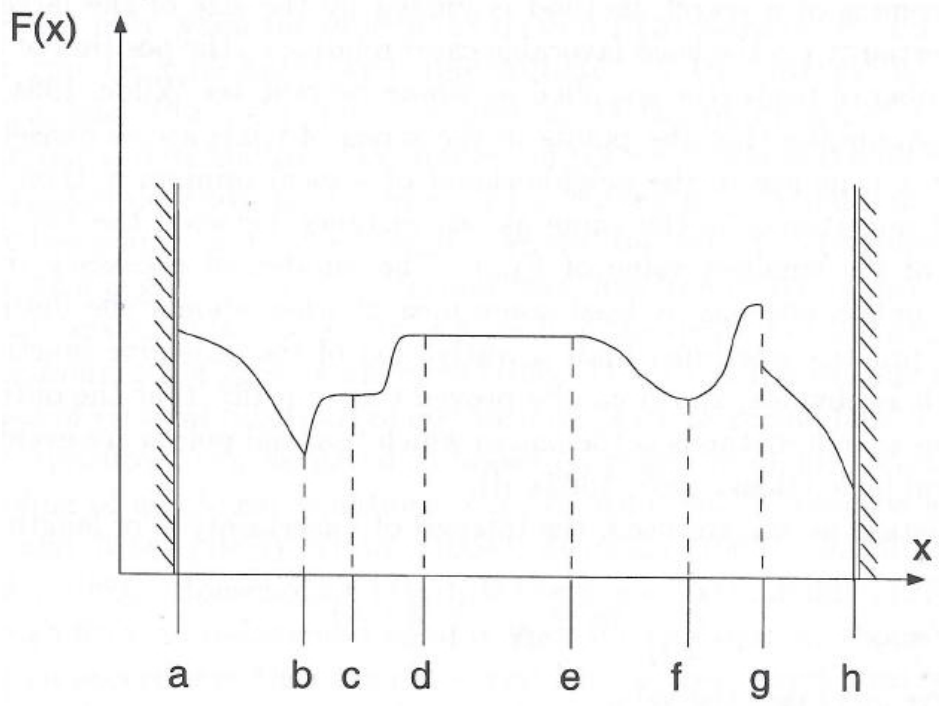



1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8


8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8

One dimensional search strategies

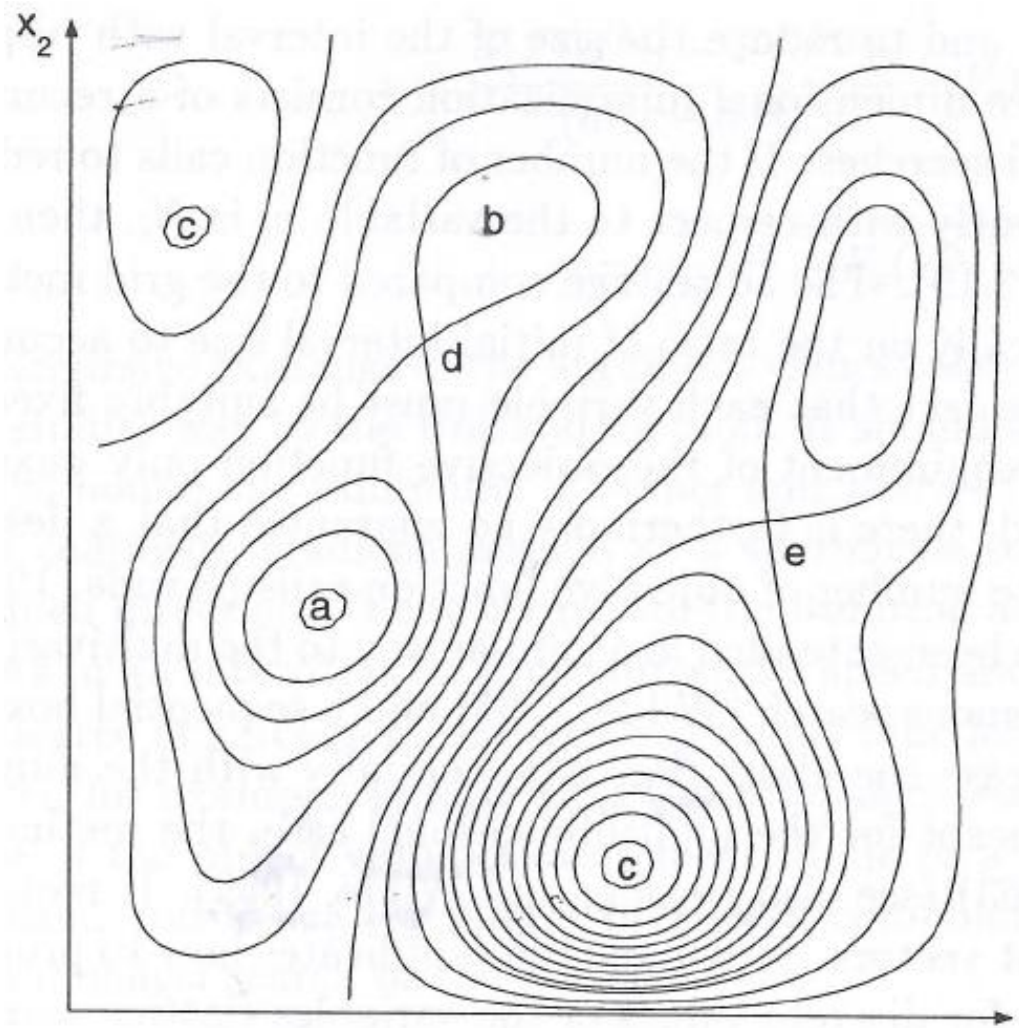
The one-dimensional search problem



- Local maxima/minima: a, b, d, e, f, g, h
- Saddle point: c
- Weak local maxima: d, e
- Global maximum: g

Multi dimensional search strategies

The multi-dimensional search problem



a: Global minimum

b: Local minimum

c: Local maxima

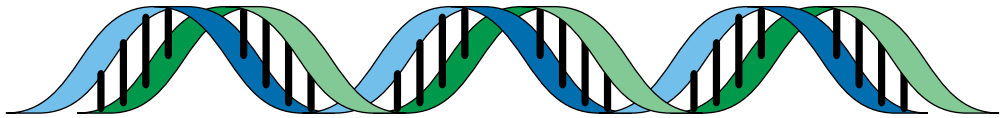
d,e: Saddle points

Stochastic methods

When problem structure is not well known, it might be hard to design appropriate deterministic search methods

Solution: randomised search approaches

Search space spanned by possible configurations for all parameters



Solutions found are not necessarily optimal

Outline

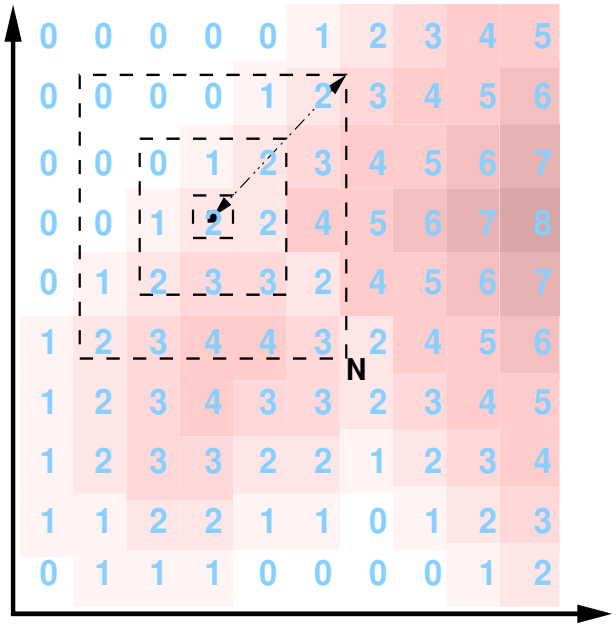
Local random search heuristics

- Local random search
  - Intuitive way to climb a mountain (by a sightless climber)

Local random search

$\forall x$ in search space S, define non-empty neighbourhood $N(x) \subseteq S$
Iteratively draw one random sample $x' \in N(x)$.
Fitness improved $(F(x) > F(x')) \Rightarrow$ new best search point.
Otherwise $\Rightarrow$ discarded.

Local random search heuristics



$N(x) = x$ or $N(x) = S$ valid, but original idea is that $N(x)$ is small set of search points.

Points $x' \in N(x)$ expected nearer to x than points $x'' \notin N(x)$

Typically, $x \in N(x)$

Local random search heuristics

Complexity reduction by restriction of the search space size

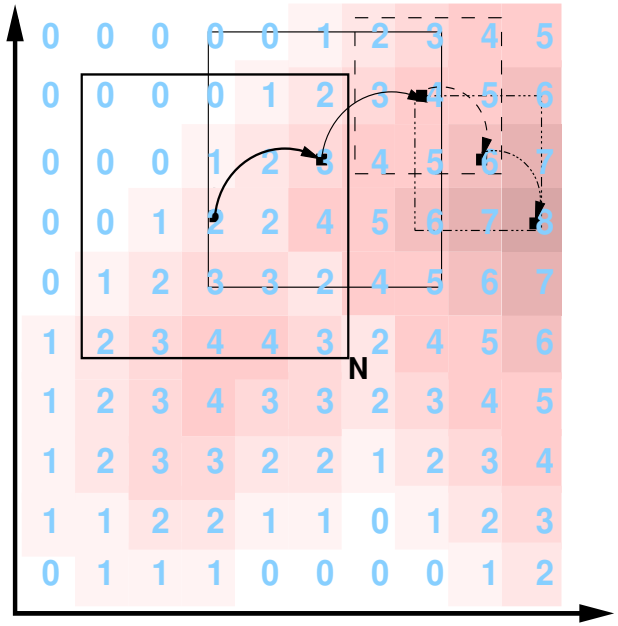
Example: $S = \{0,1\}^n$ and $N_d(x)$ are all points y with Hamming distance smaller than d ($H(x,y) \leq d$)

- For constant d we obtain: $|N_d(x)| = \Theta(n^d) \ll |S| = 2^n$

$d \leq 1$	$d \leq 2$	$d \leq 3$
1 0 1 0	1 0 1 0	1 0 1 0
0 0 1 0	0 0 1 0	1 1 1 0
1 1 1 0	1 1 1 0	1 1 1 0
1 0 0 0	1 0 0 0	1 0 0 0
1 0 1 1	1 0 1 1	1 0 1 1
	0 1 1 0	0 1 1 0
	0 0 0 0	0 0 0 0
	0 0 1 1	0 0 1 1
	1 1 0 0	1 1 0 0
	1 1 1 1	1 1 1 1
	1 0 0 1	1 0 0 1
		0 1 0 0
		0 1 1 1
		1 1 0 1
		1 0 0 1

$$|N_d(x)| = \binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{1} + \binom{n}{0}$$

Local random search heuristics



Small neighbourhood: Fast conversion to local optima

Large neighbourhood: Similar to random search

Variable neighbourhood :

- Initially, big neighbourhood, then decrease
- Challenge: Decrease not too fast

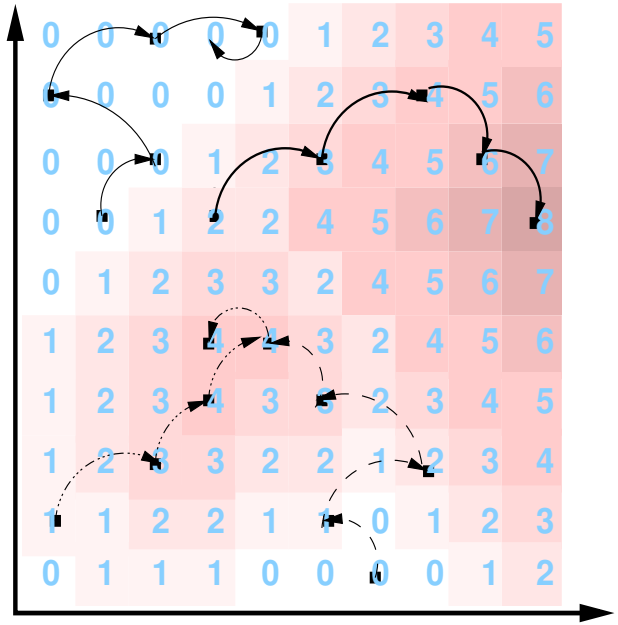
Local random search heuristics

Local optima avoidance: Multistart

- Search applied t times on problem
- Probability amplification : respectable result also with low success probability

  Assume: success probability $\delta > 0$ for one iteration

  After t iterations overall success probability: $1 - (1 - \delta)^t$



Outline

Metropolis algorithms

Local random search: only multistart can avoid local optima.

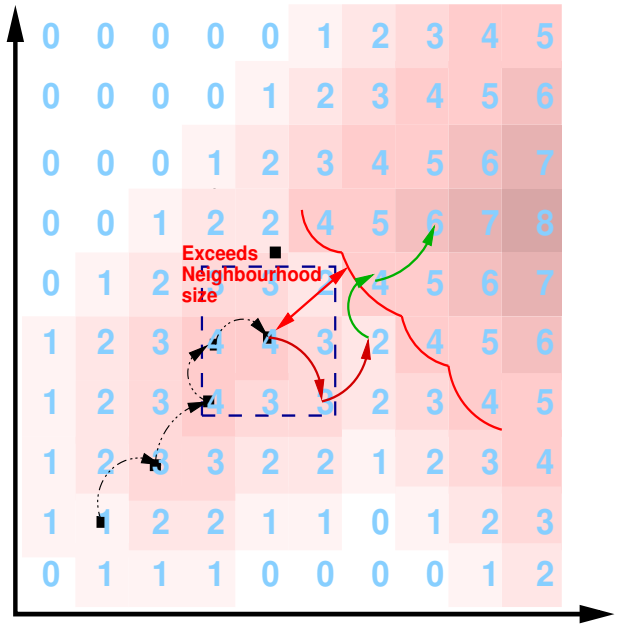
Metropolis approach accepts also search points that decrease fitness value

$F(x') > F(x) \Rightarrow x'$ discarded with prob.

$$1 - \frac{1}{e^{(F(x')-F(x))/T}}$$

$T \to 0$ random search

$T \to \infty$ uncontrolled local search



Outline

Simulated annealing

Choice of optimal T not easy $\Rightarrow$ Change during optimisation

Initially: T should allow to 'jump' to other regions of the search space with increased fitness value

Finally: Process should gradually 'freeze' until local search approach propagates the local optimum in the neighbourhood

Analogy to natural cooling processes in the creation of crystals:

- Temperature gradually decreased so that Molecules that could move freely at the beginning are slowly put into their place

Simulated annealing

No natural problem known for which it has been proved that Simulated Annealing is sufficiently more effective than the Metropolis algorithm with optimum stationary temperature

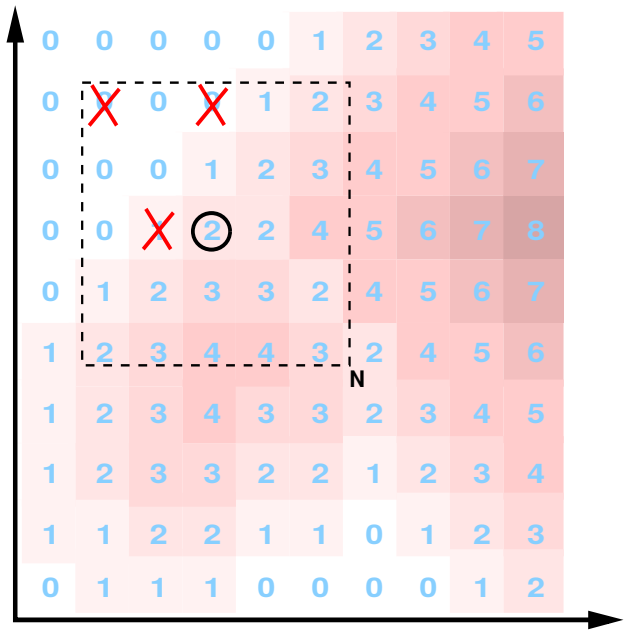
Artificially constructed problems exist, for which this could be shown

Outline

Tabu search

Algorithms discussed so far only store the actual search point

Sim. Annealing/Metropolis Search point with the best fitness value achieved so far is stored typically.



Tabu search

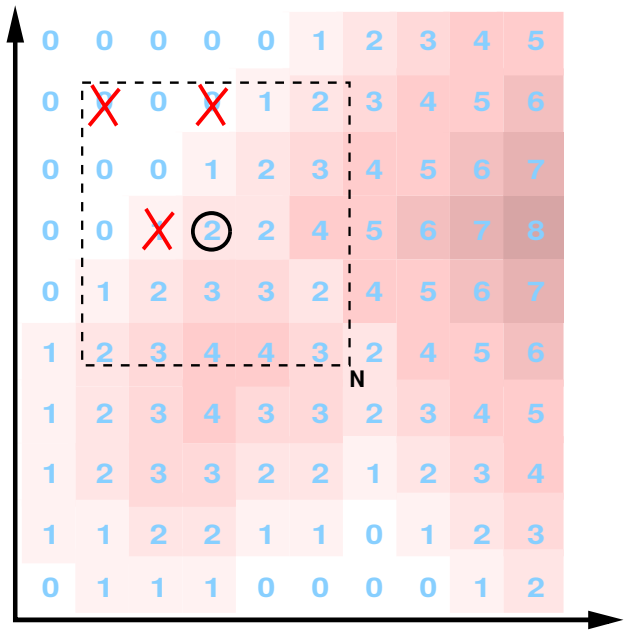
Algorithms discussed so far only store the actual search point

Sim. Annealing/Metropolis Search point with the best fitness value achieved so far is stored typically.

Other points Knowledge about all other points is typically lost

Algorithms might therefore access suboptimal points several times

$\Rightarrow$ Increased optimisation time

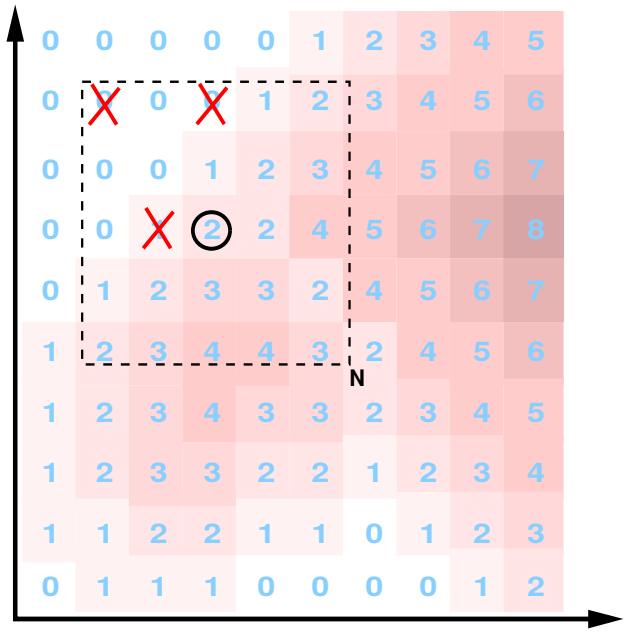


Tabu search

Tabu Search Store list of search points that have recently been accessed.

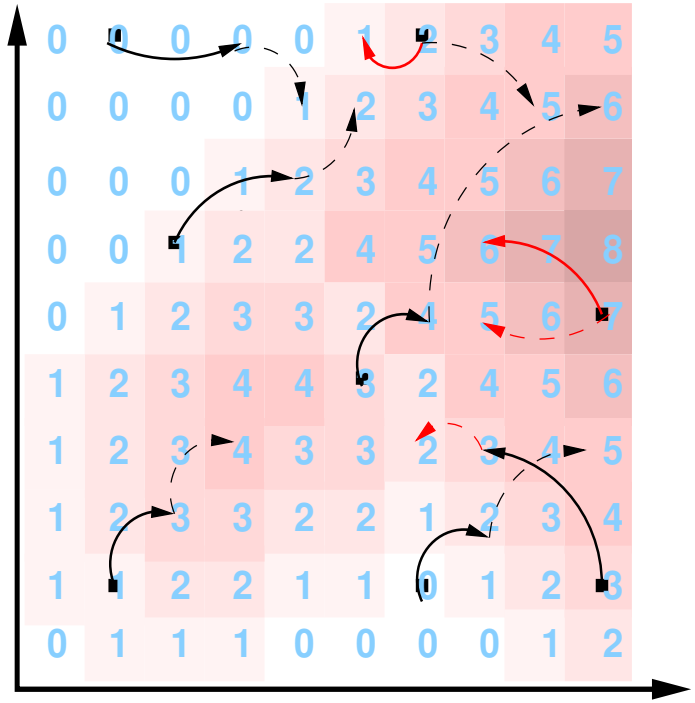
Due to memory restrictions the list is typically of finite length

When at least size of the neighbourhood $N(x)$ covered, terminate when the all points visited.



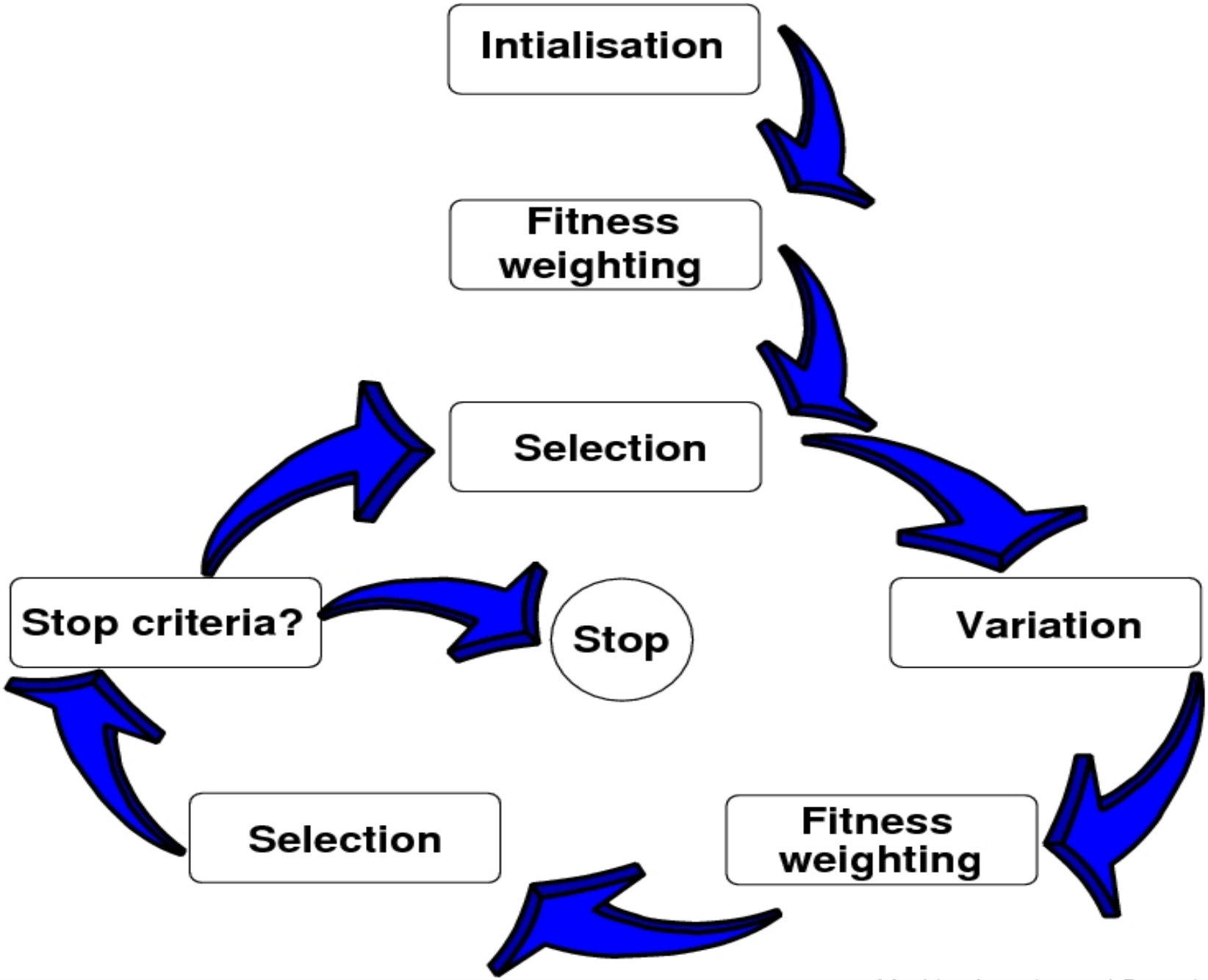
Outline

Evolutionary algorithms



Utilise evolution principles for optimisation purposes

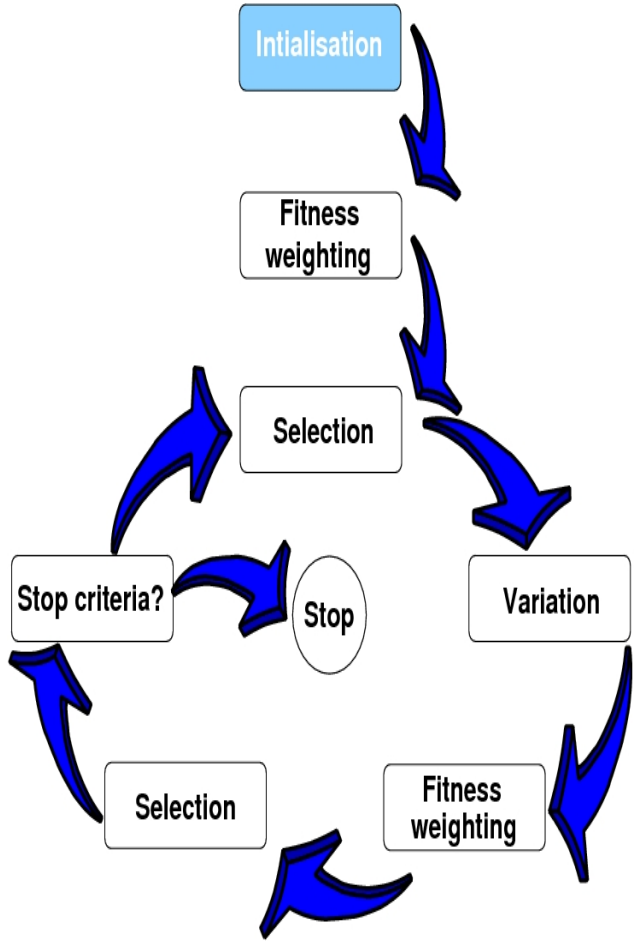
Evolutionary algorithms combine Genetic algorithms, Evolution strategies, Evolutionary programming and Genetic programming


Intialisation
Fitness weighting
Selection
Variation
Fitness weighting
Selection
Stop criteria?
Stop

Evolutionary algorithms

Initialisation

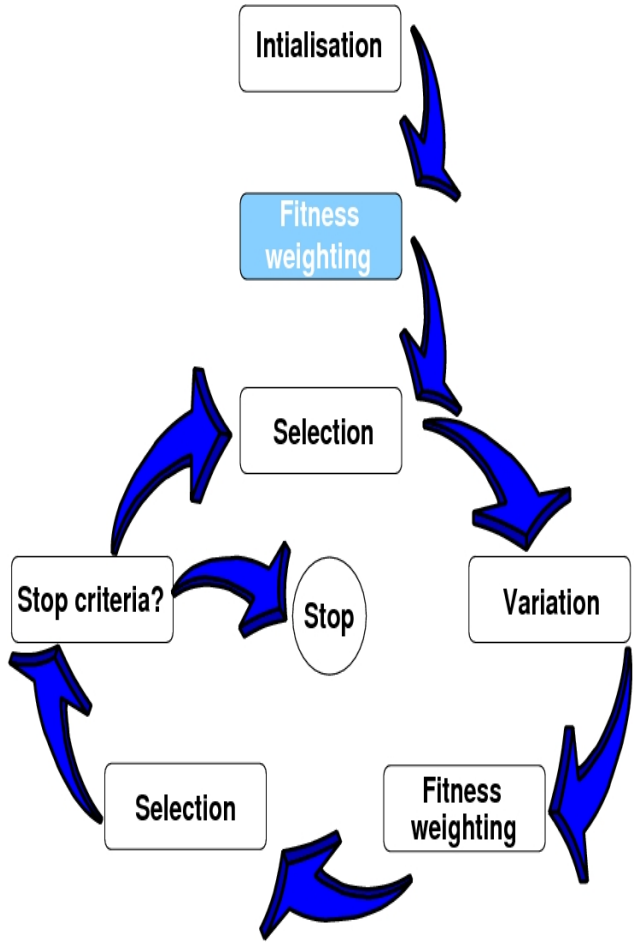
- Initialise μ individuals from the search space S
- Typically uniformly at random
- Typical search spaces: $S = \mathbb{R}^n$ or $S = \mathbb{B}^n$
- Achieve sufficient coverage:
  - Distance measure d
  - distance $\geq d$
- Improve optimisation time and quality of solution:
  - fast heuristics for individual population



Evolutionary algorithms

Fitness weighting of the population

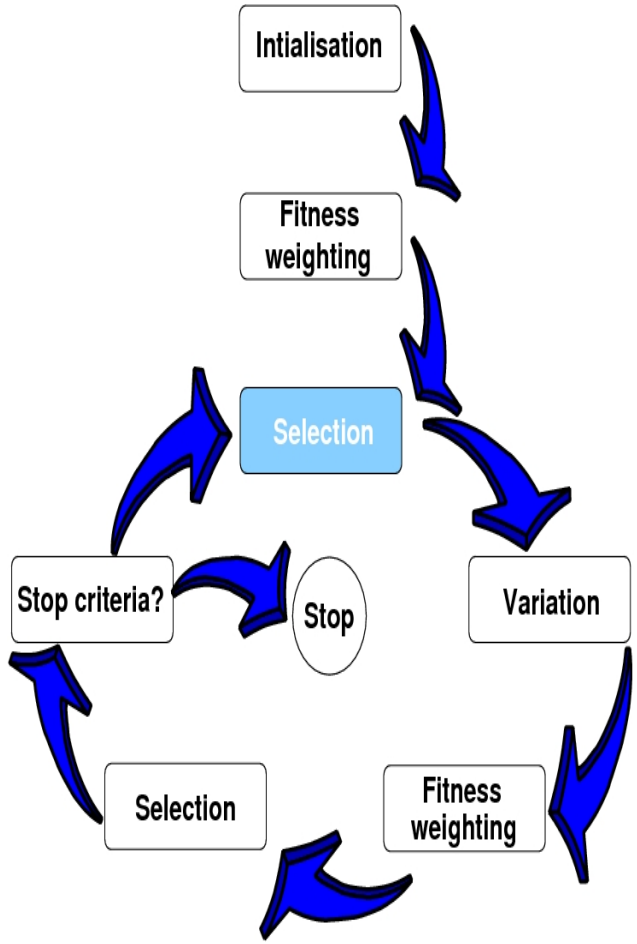
- Individuals of population weighted for their fitness value.
- Fitness function $f : S \rightarrow \mathbb{R}$
- Monotonous function



Evolutionary algorithms

Selection for reproduction

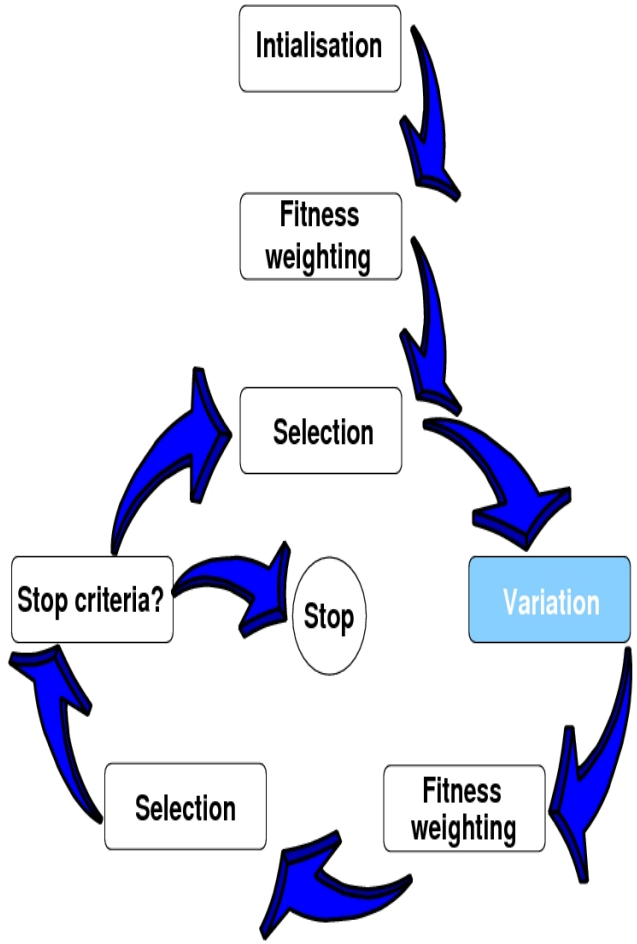
- Dependent on fitness values reached by individuals
- individuals chosen to produce offspring population
- Intuition:
  - Individuals with good fitness value: Higher probability to produce high-rated individuals for offspring population



Evolutionary algorithms

Variation

- Offspring population created by mutation and/or crossover.
- Mutation is typically local search operator
- Crossover allows to find search points in currently not populated regions
- Adaptive implementations possible



Evolutionary algorithms

Mutation

- Produces individuals that differ only slightly from the parent-individuals.
- One parent individual produces one offspring individual
- Mutation operators differ between search spaces.

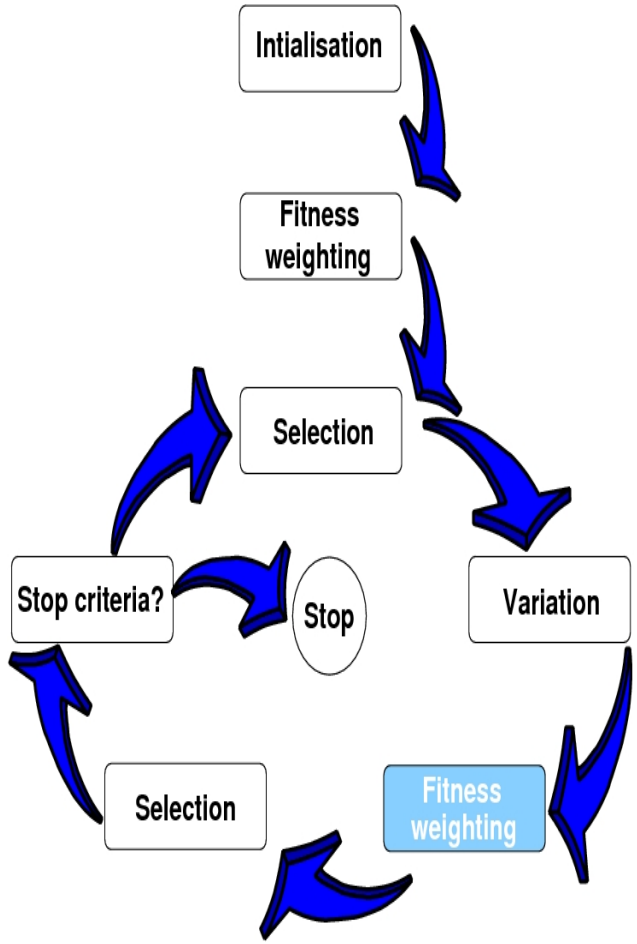
Evolutionary algorithms

Crossover

Crossover is a variation technique that produces one or more offspring individuals from two or more parent individuals

Evolutionary algorithms

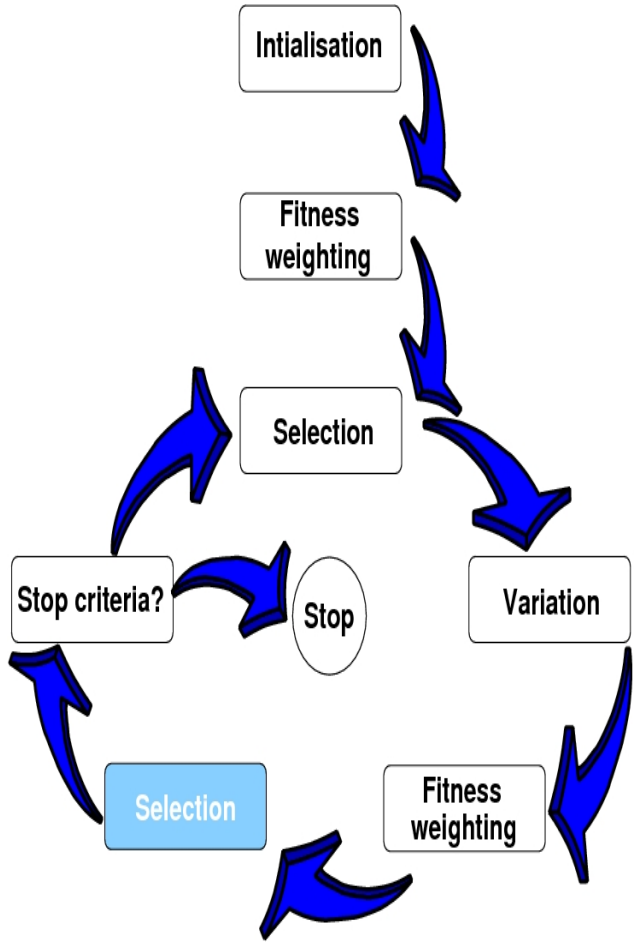
- All newly generated offspring individuals are weighted by a fitness function f.
- Structure of f impacts performance of random search approach
  - Weak multimodal vs. strong multimodal



Evolutionary algorithms

Selection for substitution

- Population size increased due to variation
- Reduce population size to μ
- Typically: Individuals with higher fitness values more probable



Evolutionary algorithms

+ and , strageties

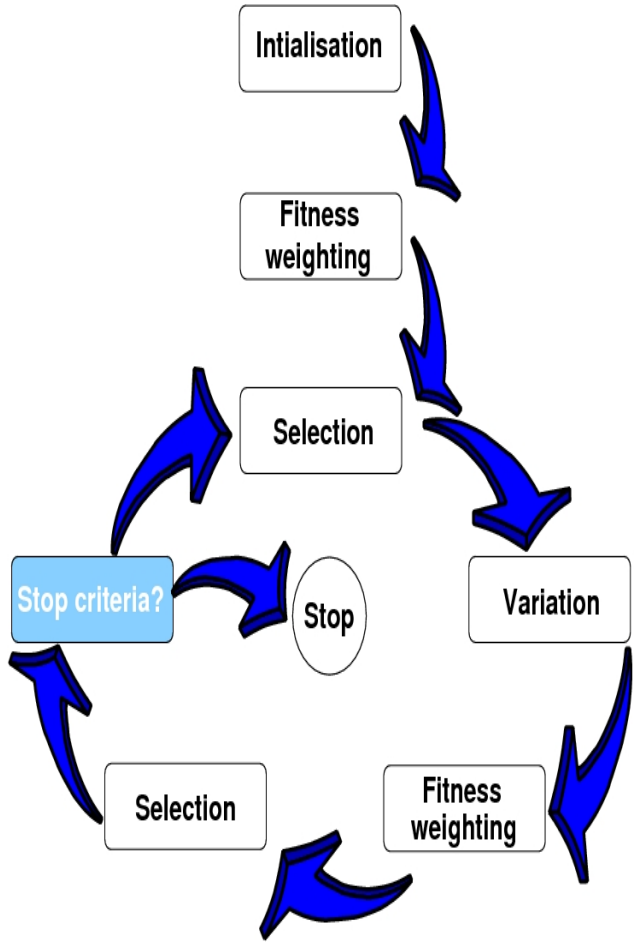
$(\mu + \lambda)$ strategies: Offspring population chosen from μ old individuals '+' λ offspring individuals

(μ, λ) strategies: μ individuals drawn from λ offspring individuals while μ old individuals are discarded

- Comma-strageties try to avoid local optima

Evolutionary algorithms

- Since global optimum not known, stop criteria required



Evolutionary algorithms

History – Genetic algorithms

- Proposed by John Holland [1]
- Binary discrete search spaces: $\{0,1\}^n$
- Fitnessproportional selection
  - For m individuals $x_1, \ldots, x_m$ the probability to choose x_i is $\frac{f(x_i)}{f(x_1)+\cdots+f(x_m)}$.
- Main evolution operator is crossover
  - Originally One-point crossover
- The main goal was not optimisation but the adaptation of an environment

[1] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

Evolutionary algorithms

History – Genetic algorithms

- The hope associated with genetic algorithms was that they are able to solve some functions especially well

Separable function

A function is called separable, if the input variables can be divided into disjoint sets $X_1, \ldots, X_k$ with $f(x) = f_1(X_1) + \cdots + f_k(X_k)$

- Since genetic algorithms utilise crossover, it was expected that they are therefore well suited to quickly find the optimum on separable functions

Evolutionary algorithms

History – Genetic algorithms

Royal road functions

k blocks of variables of length l are formed. On each block X_l the identical function f_l is implemented with

$$f_l(X_l) = \begin{cases} 1 & \text{All variables in } X_l \text{ equal 1} \\ 0 & \textit{else}. \end{cases}$$

- It was shown that genetic algorithms do NOT perform well on these functions.[2]
- The reason is that it is highly unlikely to perform crossover exactly at the border of the variable blocks.
- It is better to optimise the single blocks on their own separately by mutation.

[2] T. Jansen and I. Wegener, *Real royal road functions – where crossover provably is essential*, Discrete applied

Evolutionary algorithms

History – Evolution strategies

- Proposed by Bienert, Rechenberg and Schwefel[3] [4]
- At first only steady search spaces as $\mathbb{R}^n$
- No Crossover
- Only mutation
  - First mutation operator: Each component x_i is replaced by $x_i + Z_i$ (Z_i normally distributed, σ^2 Variance)

[3] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, 1973.*

[4] H.P. Schwefel, *Evolution and optimum seeking*, 1993

Evolutionary algorithms

History – Evolution strategies

1/5 rule

After $10n$ iterations, the variance is adopted every n iterations. When the number of accepted mutations in the last $10n$ steps is greater than $1/5$, σ is divided by 0.85 and else multiplied by 0.85.

- This heuristic is based on an analysis of the fitness function $x_1^2, \ldots, x_n^2$ – the sphere model.

Evolutionary algorithms

History – Evolutionary programming

- The approach was proposed by Lawrence J. Fogel[5][6]
- Various similarities to evolution strategies
- Search Space: Space of deterministic finite automata.

[5] L.J. Fogel, *Autonomous automata*, Industrial Research, Vol. 4, 1962.

[6] L.J. Fogel *Biotechnology: Concepts and Applications*, Prentice-Hall, 1963

Evolutionary algorithms

History – Genetic programming

- Proposed by John Koza[7]
- Search space: Syntactically correct programs
- Crossover more important than mutation

[7] John Koza *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992

Evolutionary algorithms

- Since evolutionary approaches are typically slow to initially find a search point with a reasonable fitness value,
- Approaches are combined with fast heuristics that initially search for a good starting point.
- Afterwards the evolutionary approach is applied

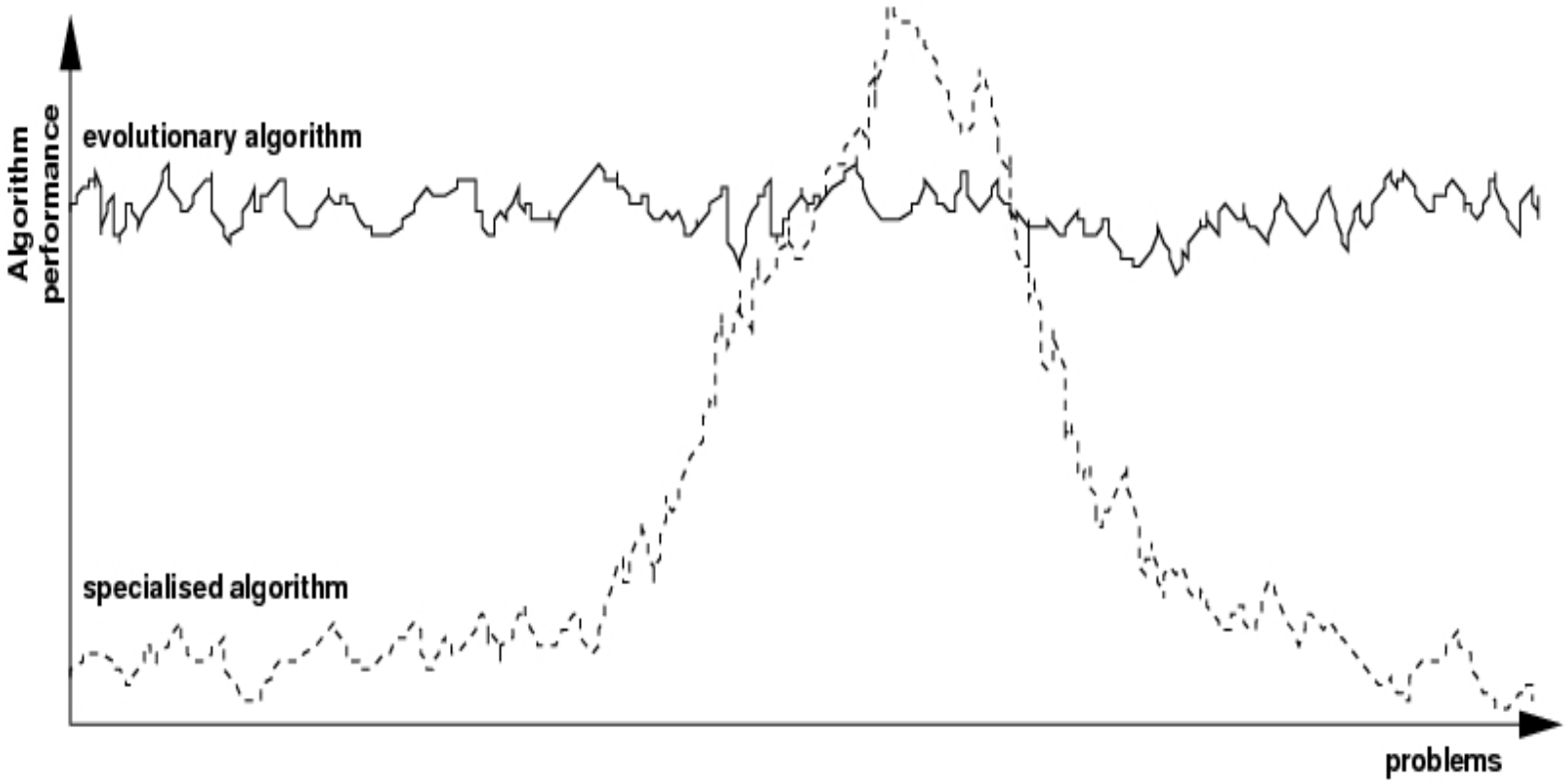
Limitations of evolutionary approaches

It has been argued that

- Problem specific algorithms better than evolutionary on small subset of problems
- Evolutionary algorithms better on average over all problems

Evolutionary algorithms proposed as general purpose optimisation scheme

Limitations of evolutionary approaches



Limitations of evolutionary approaches

Can an algorithm be suited for 'all' problems?

What does 'all problems' mean?

Can one algorithm be better on average than another algorithm on 'all' problems?

Limitations of evolutionary approaches

Can an algorithm be suited for 'all' problems?

- Distinct coding of the search space
- Various fitness functions

What does 'all problems' mean?

Can one algorithm be better on average than another algorithm on 'all' problems?

Limitations of evolutionary approaches

Can an algorithm be suited for 'all' problems?

- Distinct coding of the search space
- Various fitness functions

What does 'all problems' mean?

- All possible representations and sizes of search space
- All possible fitness functions
- Every single point is the optimum point in several of these problems

Can one algorithm be better on average than another algorithm on 'all' problems?

Limitations of evolutionary approaches

Wolpert and Macready formalised this assertion:[8]

F Set of all functions $f : S \rightarrow W$

- S and W finite (every computation on physical computers only has finite resources)
- Fitness function evaluated only once per search point

$A(f)$ number of points evaluated until optimum is found

[8] D.H. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimisation*, IEEE Transactions on Evolutionary Computation 1, 67, 1997.

Limitations of evolutionary approaches

No free lunch theorem

Assume that the average performance of an algorithm in the No Free Lunch Scenario for S and W is given by $A_{S,W}$, the average over all $A(f), f \in F$. Given two algorithms A and A', we obtain $A_{S,W} = A'_{S,W}$

- This means that two arbitrary algorithms perform equally well on average on all problems

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

W.l.o.g.: $W = \{1, \ldots, N\}$
We consider sets $F_{s,i,N}$ of all functions f on a search space of non-visited search points of size s with at least one x with $f(x) > i$
Observe that for every function f and every permutation π also f_π belongs to $F_{s,i,N}$

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

Proof by induction over $s := |S|$.
Induction start: $s = 1$

Every algorithm has to choose the single optimum search point with its first request.

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$
Define $a : S \rightarrow \mathbb{N}$ so that $\forall x \in S$ the share of functions with $f(x) = j$ is exactly $a(j)$.

This is independent of x, since all permutations f_π of a function f also belong to $F_{s,i,N}$

$a(j)$ is therefore the probability to choose a search point with fitness value j (Independent of the concrete algorithm A)

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

Count of functions $f(x) = j$ is equal to the number of functions $f_\pi(y) = j$, since all permutations of f are also in $F_{s,i,N}$.

The probability to achieve a fitness value $j > i$ is therefore independent of the algorithm.

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

If $j \leq i$, x is not optimal in scenario $F_{s,i,N}$ and the new scenario is $F_{s-1,i,N}$

Limitations of evolutionary approaches

Proof of the No Free Lunch Theorem

Summary – in other words:

For any two algorithms we can state a suitable permutation of the Problem-function for one problem (i.e. state another problem), so that both algorithms in each iteration request identical search points.

- Especially, since every search point could be optimal, there are always algorithms that request the optimal search point right from the start.

Limitations of evolutionary approaches

NFL is possible, since ALL algorithms and ALL problems are considered

Is there an NFL valid in smaller, more realistic scenarios?

In [9] a similar theorem was proved for more realistic problem scenarios.

[9] S. Droste, T. Jansen and I. Wegener, *Perhaps not a free lunch but at least a free appetizer*, Proceedings of the 1st Genetic and Evolutionary Computation Conference, 1999.

Design aspects of evolutionary algorithms

Selection principles

Uniform selection

Individuals chosen uniformly at random

Deterministic selection

Deterministically choose the highest rated individuals for the selection

Threshold selection

Candidates for offspring population drawn uniformly at random from the t highest rated individuals

Design aspects of evolutionary algorithms

Selection principles

Fitnessproportional selection

- For population $x_i, \ldots, x_n$ individual x_i chosen with

$$p(x_i) = \frac{f(x_i)}{f(x_1) + \cdots + f(x_n)}$$

- Draw random variable u from $[0, 1]$ and consider x_i if

$$p(x_1) + \cdots + p(x_{i-1}) < u \leq p(x_1) + \cdots + p(x_i)$$

- Frequently applied for evolutionary approaches

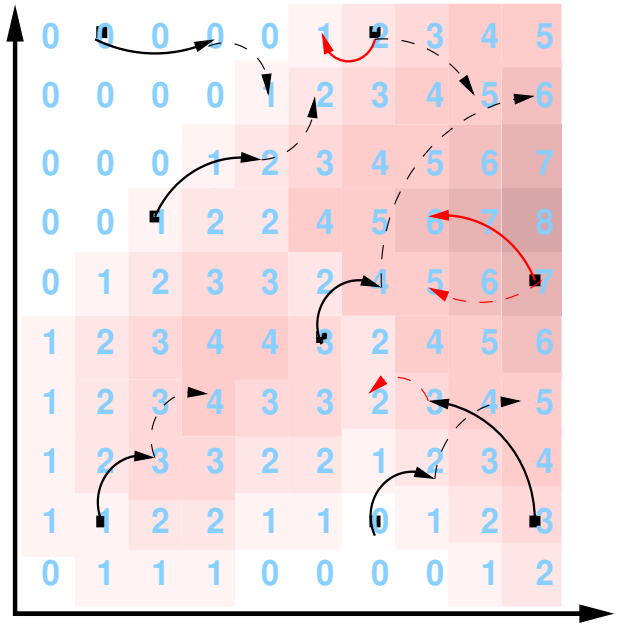
Design aspects of evolutionary algorithms

Selection principles

- Problems with Fitnessproportional selection
  - Linear modification of the fitness function ($f \rightarrow f + c$) results in different behaviour
  - When fitness values sufficiently separated, selection is nearly deterministic
  - When deviation in fitness values is small relative to absolute values, similar to uniform selection

Design aspects of evolutionary algorithms

Variation operators – Mutation



- Mutation creates one offspring individual from one individual
- Operators are designed for specific search spaces
- Shall apply only few modifications of individuals on average
- Distant individuals have smaller probability

Evolutionary algorithms

Variation operators – Mutation

Mutation operators for individuals from $\mathbb{B}^n$ (similar operators for other search spaces):

Standard bit mutation

- Offspring individual created bit-wise from parent individual
- Every bit 'flipped' with probability p_m
- Common choice: $p_m = \frac{1}{n}$

1 bit mutation

- Offspring individual identical in all but one bit.
- This bit chosen uniformly at random from all n bits

Design aspects of evolutionary algorithms

Variation operators – Crossover

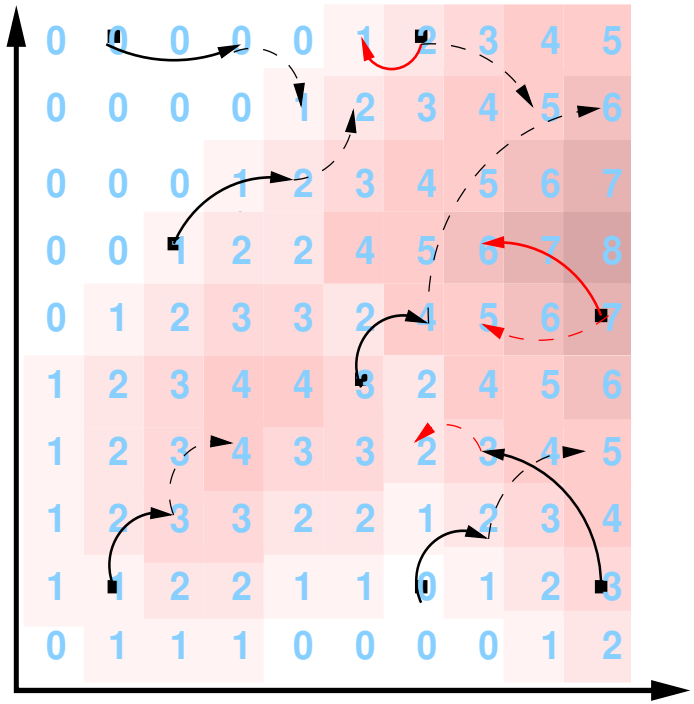
- Crossover typically takes two individuals and results in one or two offspring individuals
  - Also crossover of more than two individuals possible
  - Often generalisations of the two-individual case
- Distinct crossover methods for various search spaces
- Crossover parameter p_c specifies the probability with which crossover (and not mutation) is applied for one selected individual
- In some cases (e.g. binary coded numbers) not all positions in the individual string are allowed to apply crossover on

Design aspects of evolutionary algorithms

Variation operators – Crossover

Typical crossover variants

- One-point crossover
- k-point crossover
- Uniform crossover



Evolutionary algorithms

Variation operators – Crossover operators for $\mathbb{B}^n$ (Other search spaces similar)

One-point crossover: Individual x'' from two individuals x and x' according to uniformly determined crossover position:

$$x''_j = \begin{cases} x_j & \text{if } j \leq i \\ x'_j & \text{if } j > i \end{cases}$$

Evolutionary algorithms

Variation operators – Crossover operators for $\mathbb{B}^n$ (Other search spaces similar)

k-point crossover: Choose $k \leq n$ positions uniformly at random:

$$
\begin{array}{rl}
x_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} | x_{1k_1+1}, \ldots, x_{1k_2} | x_{1k_2+1}, \ldots, x_{1n} \\
x_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} | x_{2k_1+1}, \ldots, x_{2k_2} | x_{2k_2+1}, \ldots, x_{2n} \\
\hline
y_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} | x_{2k_1+1}, \ldots, x_{2k_2} | x_{1k_2+1}, \ldots, x_{1n} \\
y_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} | x_{1k_1+1}, \ldots, x_{1k_2} | x_{2k_2+1}, \ldots, x_{2n}
\end{array}
$$

Evolutionary algorithms

Variation operators – Crossover operators for $\mathbb{B}^n$ (Other search spaces similar)

Uniform crossover: Each bit chosen with uniform probability from one of the parent individuals

Design aspects of evolutionary algorithms

Discussion

Easy implementation EAs are easy to implement when compared to some specialised approaches

Computationally complex However, EAs are computationally complex

$\Rightarrow$ It is therefore beneficial to implement efficient variants to the distinct methods

Design aspects of evolutionary algorithms

Discussion

Pseudo random bits Generation of PRB important for many of the theoretic results for EAs to hold

Reduce random experiments More efficient to calculate the next flipping bit in a mutation instead of doing the calculation for every bit independently

Design aspects of evolutionary algorithms

Discussion

Fitness value calculation Most of the computational time is typically consumed by the fitness calculation

Prevent re-calculation for individuals Dynamic data structures that support search and insert

Outline

Questions?

Stephan Sigg
stephan.sigg@cs.uni-goettingen.de

Literature

- C.M. Bishop: Pattern recognition and machine learning, Springer, 2007.
- R.O. Duda, P.E. Hart, D.G. Stork: Pattern Classification, Wiley, 2001.