Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering
and Networks Laboratory

Bio-inspired Optimization and Design

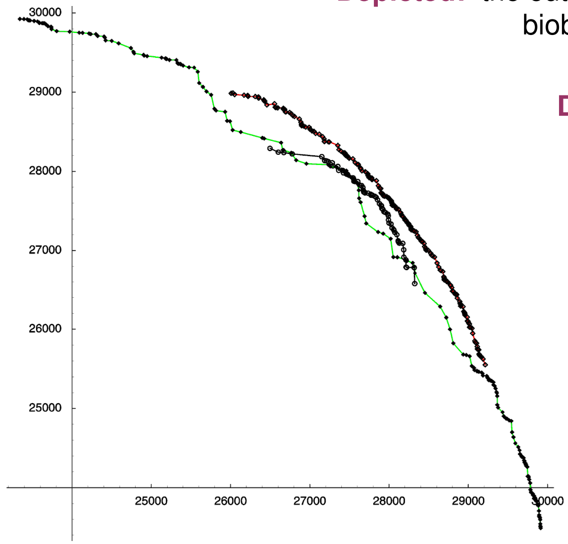
Eckart Zitzler

5. Performance Assessment

Performance Assessment in a Nutshell

Depicted: the outcomes of 3 algorithms for a biobjecive knapsack problem...

Decide: which one provides the best performance?



In the Following...

...you learn:

- what aspects performance assessment includes;
- why there is no general best performing randomized search algorithm;
- on the basis of one example what type of hard statements can be made about the performance of randomized search algorithms.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering
and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

5. Performance Assessment

What Is Performance?

performance = quality of the outcome / time resources invested

Issues:

- How to measure time?
  - overall execution time (OS, programming languages flaws)
  - number of objective function evaluations
- How to measure quality?
  - single objective: objective function value
  - multiple objectives: what is the quality of a Pareto set approximation?
- How to take randomness and parameterization into account?
  - influence of the initial population, parameters, etc.
  - statistics: expected value, variance, etc.
- How to choose the benchmark problems?
  - simple to implement, but representative for complex applications
  - how many is enough?

Performance Assessment: Approaches

Which technique is suited for which problem class?

❶ **Theoretically (by analysis):** difficult

- Limit behavior (unlimited run-time resources):
  does the algorithm converge to the optimum when run for an infinite number of iterations?
- Running time analysis:
  what is the expected number of objective function evaluations in the worst / average / best case?

❷ **Empirically (by simulation):** standard

- using standard parameter settings
- multiple runs, e.g., 30, for each algorithm under consideration
- statistical testing procedure for comparing sets of outcomes

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering
and Networks Laboratory

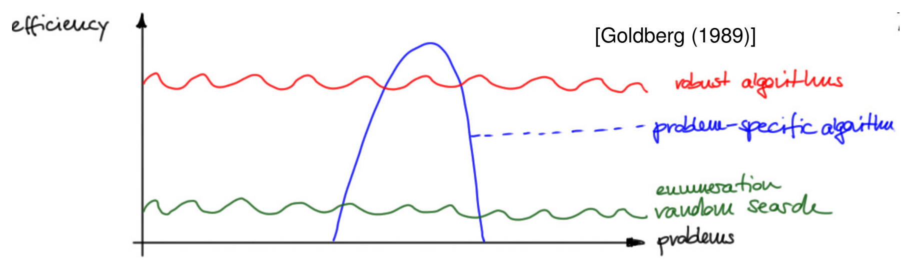
Bio-inspired Optimization and Design

Eckart Zitzler

5. Performance Assessment

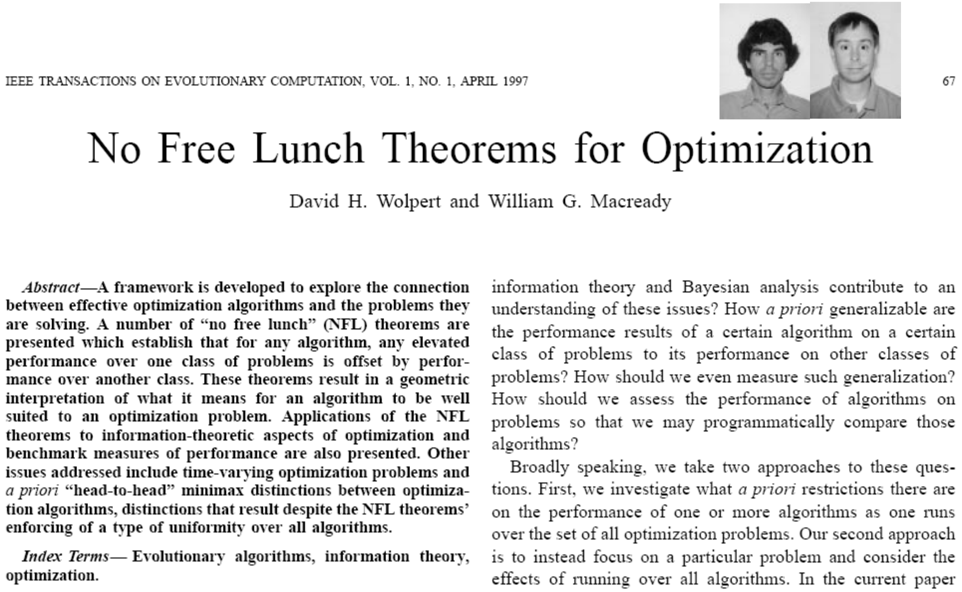
The Assumption: Some Are Better Than Others...

- various variants of randomized search have been proposed
- are some more robust (more efficient) than others?



- assumption behind this figure: some algorithms are better than others in average
- in the following, we will see that this assumption does not hold in general

The Bad News: In General This Is Not True...



IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, APRIL 1997 67

No Free Lunch Theorems for Optimization

David H. Wolpert and William G. Macready

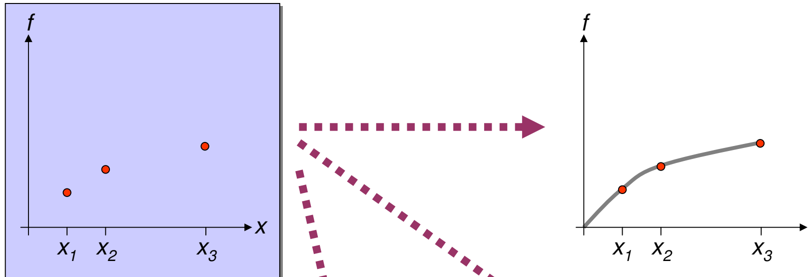
Abstract—**A framework is developed to explore the connection between effective optimization algorithms and the problems they are solving. A number of "no free lunch" (NFL) theorems are presented which establish that for any algorithm, any elevated performance over one class of problems is offset by performance over another class. These theorems result in a geometric interpretation of what it means for an algorithm to be well suited to an optimization problem. Applications of the NFL theorems to information-theoretic aspects of optimization and benchmark measures of performance are also presented. Other issues addressed include time-varying optimization problems and *a priori* "head-to-head" minimax distinctions between optimization algorithms, distinctions that result despite the NFL theorems' enforcing of a type of uniformity over all algorithms.**

Index Terms— **Evolutionary algorithms, information theory, optimization.**

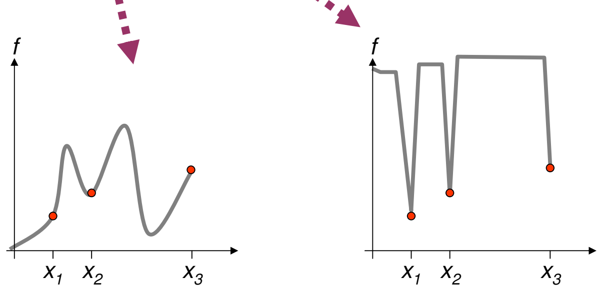
information theory and Bayesian analysis contribute to an understanding of these issues? How *a priori* generalizable are the performance results of a certain algorithm on a certain class of problems to its performance on other classes of problems? How should we even measure such generalization? How should we assess the performance of algorithms on problems so that we may programmatically compare those algorithms?

Broadly speaking, we take two approaches to these questions. First, we investigate what *a priori* restrictions there are on the performance of one or more algorithms as one runs over the set of all optimization problems. Our second approach is to instead focus on a particular problem and consider the effects of running over all algorithms. In the current paper

No-Free-Lunch-Theorem: Idea



Given three solutions and their objective values, do they tell you which function is hidden behind?



The Set Of Problems

Question: What optimization problems do we consider?

- Only single-objective problems will be considered
- The decision space is finite
- Without loss of generality, $X \subset \aleph$, $Z = \aleph$
- A maximization problem is assumed

$\Rightarrow$ The set of considered problems for a given decision space $X \subset \aleph$ is described by all functions $f: X \rightarrow \aleph$, each representing another optimization problem $(X, \aleph, f, \geq)$:

$$F_X := \{f \mid f : X \rightarrow \aleph\}$$

Note: F_X is closed under permutations, i.e., for any $f \in F_X$ and permutation $\pi: \aleph \rightarrow \aleph$ is also the function $f_\pi(x) := f(\pi(x))$ in F_X

One Notion of Performance

Question: What is the performance of a randomized search algorithm in a black-box scenario?

Now, given an arbitrary, finite function f, how can we define the performance or efficiency of an algorithm A on f? There are different possibilities, e.g., we could consider the running-time complexity of A for generating an optimal solution for f — however, this is often difficult. Instead, we define performance as

$C_{A,f}$ = expected number of different solutions visited until an optimal solution is found

This definition assumes that every algorithm never generates previously visited solutions again. This can be achieved by keeping all visited solutions in the internal memory of the search algorithm. From a practical point of view, though, this is questionable. Nevertheless, for a theoretical investigation it makes life much easier. The same holds for the assumption that the algorithms knows when an optimal solution has been found.

One Notion of Performance (Cont'd)

If we consider now two deterministic search algorithms, they differ with respect to f simply in the order according to which the solutions in X are visited:

$C_{A,f} = 4$

$C_{B,f} = 5$

If we consider a randomized search algorithm, the sequence of visited solutions is likely to be different for each run of the algorithm. Therefore, $C_{A,f}$ denotes the expected number of visited solutions.

For a set F of functions, we define $C_{A,F}$ as follows:

$$C_{A,F} = \frac{1}{|F|} \sum_{f \in F} C_{A,f}$$

$C_{A,F}$ denotes the average $C_{A,f}$-value for $f \in F$.

The No-Free-Lunch Scenario / Theorem

No-Free-Lunch Scenario

A = algorithm generating a permutation over X

F = set of functions $f: X \rightarrow \mathbb{N}$ with $|X| \in \mathbb{N}$, $X \subset \mathbb{N}$

$C_{A,F}$ = average number of visited solutions

Note that it is a black-box optimization scenario, i.e., nothing is known about f.

No-Free-Lunch Theorem (NFL)

In the no-free-lunch scenario, for any two algorithms A and B it holds

$C_{A,F} = C_{B,F}$

if all $f \in F$ are equally likely.

Illustration of the Proof

The idea of the proof is quite simple: we show that for any solution in the search space the probability of being optimal is the same for all solutions, if we average over all possible functions.

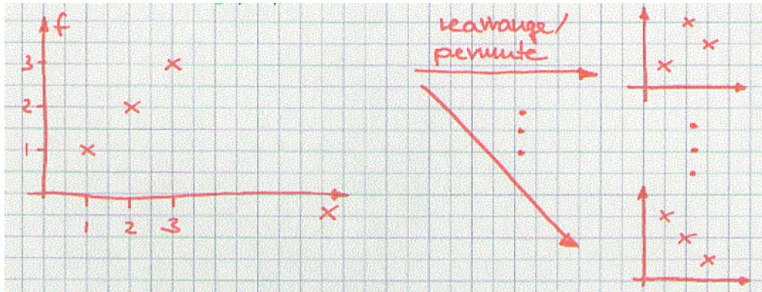
Example

$X = \{1,2,3\}$

$F_X^* = \{ f : X \rightarrow \{1,2,3\} \mid f(1) \neq f(2) \neq f(3) \}$

That means F_X^* contains all bijective (one-to-one) functions over the set $\{1,2,3\}$.

Illustration of the Proof (Cont'd)



F_X^* represented as a matrix

x	1	2	3
f(x)	1	2	3
	1	3	2
	2	1	3
	2	3	1
	3	1	2
	3	2	1

Illustration of the Proof (Cont'd)

Now, what is the expected number of visited solutions for an algorithm A?

$C_{A, F^*_X} = ?$

We consider the sequence $(x_0, x_1, x_2, \dots)$ of solutions generated by A:

$\text{Prob}[f(x_1) = 3] = \frac{1}{3}$ $\qquad x_1 \in \{1, 2, 3\}$

$\text{Prob}[f(x_2) = 3 \mid f(x_1) \neq 3] = \frac{1}{2}$ $\qquad x_2, x_1 \in \{1, 2, 3\}$, $x_2 \neq x_1$

$\text{Prob}[f(x_3) = 3 \mid f(x_1) \neq 3 \wedge f(x_2) \neq 3] = 1$ $\qquad x_3, x_1, x_2 \in \{1, 2, 3\}$, $x_3 \neq x_1 \neq x_2$

We observe that of the actually chosen solution, the chance of finding the optimum in each step is the same for all solutions.

$$C_{A, F^*_X} = 1 \cdot \frac{1}{3} + 2 \cdot \left(1 - \frac{1}{3}\right) \cdot \frac{1}{2} + 3 \cdot \left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{2}\right) \cdot 1$$
$$= 2$$

Proof of the NFL Theorem

<u>NFL proof</u>

$F_{X,i,j}$ = set of functions $f \in F$ with

1. $f : X \to \mathbb{N}$ where $|X| \in \mathbb{N}$
2. $j \in \mathbb{N}$ is the optimal function value, i.e., $f(x) \leq j$ for all $x \in X$
3. exactly i optimal solutions exist, i.e., $|\{x \in X \mid f(x) = j\}| = i$

Example: $F^*_{\{1,2,3\}} \subseteq F_{\{1,2,3\},1,3}$

This construction simplifies the proof. Note that

$$F = \dot{\bigcup_{\substack{X \subset \mathbb{N} \\ |X| \in \mathbb{N} \\ 1 \leq i \leq |X| \\ j \in \mathbb{N}}}} F_{X,i,j}$$

Proof of the NFL Theorem (Cont'd)

We will show that

$$C_{A, F_{X,i,j}} = C_{B, F_{X,i,j}}$$

for any two algorithms A and B and search space X with $|X| \in \mathbb{N}$.

The proof is via induction over $|X|$, the size of the search space.

$|X| = 1$: $\quad C_{A, F_{X,i,j}} = 1$ ✓

$|X| - 1 \to |X|$: $\quad x_0$ = first solution visited

As for $f \in F_{X,i,j}$ and any permutation π over X, also the function $f_\pi \in F_{X,i,j}$ with

$f_\pi(x) := f(\pi(x))$

it holds

$\text{Prob}[f(x) = j] = \frac{i}{|X|}$

for all $x \in X$, and therefore also for x_0.

Proof of the NFL Theorem (Cont'd)

Now, assume that x_0 is not optimal, i.e., $f(x_0) = z < j$. The remaining possible functions are

$R = \{f \in F_{X,i,j} \mid f(x_0) = z\}$

That is, we have to optimize any of the functions

$P = \{f \in F_{X \setminus \{x_0\},i,j} \mid \exists f' \in R : f(x) = f'(x) \text{ for } x \in X \setminus \{x_0\}\}$

Obviously, $P \subseteq F_{X \setminus \{x_0\},i,j}$. Also $F_{X \setminus \{x_0\},i,j} \subseteq P$ as

$f \in F_{X \setminus \{x_0\},i,j}$

$\Rightarrow f' \in R$ with $f'(x) = \begin{cases} z & \text{if } x = x_0 \\ f(x) & \text{if } x \in X \setminus \{x_0\} \end{cases}$

$\Rightarrow f \in P$

Therefore, we obtain

$$C_{A, F_{X,i,j}} = 1 \cdot \frac{i}{M} + \left(1 - \frac{i}{M}\right)\left(1 + C_{A, F_{X \setminus \{x_0\},i,j}}\right)$$

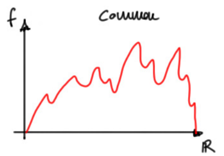
for any algorithm A. □

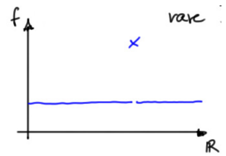
Discussion of The NFL Theorem

Question: Is the NFL scenario actually realistic?

Some reasons for criticism:

- not all possible functions f F are equally likely, some are even not computable:





- the assumption that each solution will be visited once at maximum is not realistic
- observation in practice: random search worse than, e.g., simulated annealing
- black-box assumption problematic: representation, neighborhood are problem-specific

Implications of The NFL Theorem

Does that mean the NFL theorem is useless?

No, it needs to be seen as a theoretical validation of the assumption stated in [Goldberg 1989], and theory usually needs a high abstraction level. It indicates that this assumption most likely does not hold for most realistic scenarios. Furthermore, there has not been any further work that proved the opposite.

Nevertheless, there may be classes of functions where some algorithms are better than others, and theoretical studies have been published to show this. To determine theoretically and practically which type of algorithm is best suited to which type of problem is the subject of ongoing research.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering
and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

5. Performance Assessment

Running Time Analysis of a Simple RSA

Problem: ONEMAX

$$f_{ONEMAX}(\mathbf{x}) = \sum_{i=1}^{n} x_i \text{ with } \mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$$

Algorithm:

- Evolutionary algorithm with population size *N=1*
- (1+1) environmental selection strategy
- bit flip mutation with $p_m = 1 / (n+1)$
- no recombination

Question: What is - in the worst case - the expected number of iterations that need to be performed in the evolutionary algorithm until the optimal solution has been found?

Proof

Assume that $y = (y_1, y_2, \ldots, y_n) \in \{0,1\}^n$ is the current population member and contains exactly i ones. What is the chance s_i that a mutation leads to a better solution?

$$s_i \geq \underbrace{1/n \cdot (1 - (1/n))^{n-1}}_{\text{probability that exactly one bit is flipped in the individual } y} \cdot \underbrace{(n-i)}_{\text{number of components in } y \text{ set to } 0}$$

The first multiplier can be simplified by using a commonly known formula:

$$(1/n) \cdot (1 - (1/n))^{n-1} \geq \frac{1}{e \cdot n}$$

Therefore:

$$s_i \geq \frac{n-i}{e \cdot n}$$

Proof (Continued)

Given s_i, how many mutations are necessary until a better solution has been generated? The corresponding random variable follows a geometric distribution, and therefore the expectation value is $1/s_i$. That means in average $1/s_i$ mutations are required to improve a solution with i ones.

Now, assume we start with the worst case $y = (0, 0, \ldots, 0)$. What is the expected number of mutations until the optimal solution $y^* = (1, 1, \ldots, 1)$ has been generated? It can be estimated using the upper bound

$$\sum_{i=0}^{n-1} (1/s_i) \leq \sum_{i=0}^{n-1} \frac{e \cdot n}{n-i} = e \cdot n \cdot \sum_{i=0}^{n-1} 1/(n-i) = e \cdot n \underbrace{\sum_{i=1}^{n} \frac{1}{i}}_{\text{harmonic number}}$$

$$\leq e \cdot n \cdot \ln n$$

Therefore, the expected number of iterations in the worst case until the optimum has been found is of order $O(n \log n)$.

Running Time Analysis: A Biobjective Scenario

Problem:

maximize leading ones (f_1), trailing zeros (f_2)

1	1	0	1	0	0	0

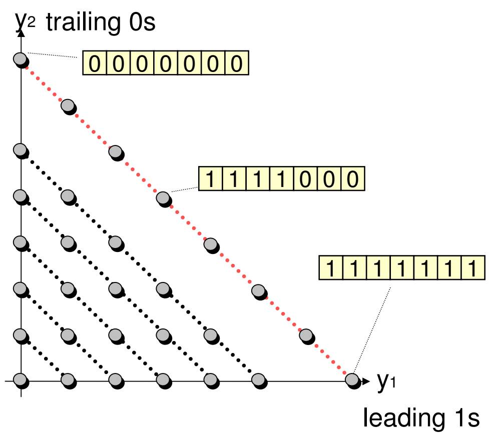
Variation:

single point mutation

1	1	0	1	0	0	0

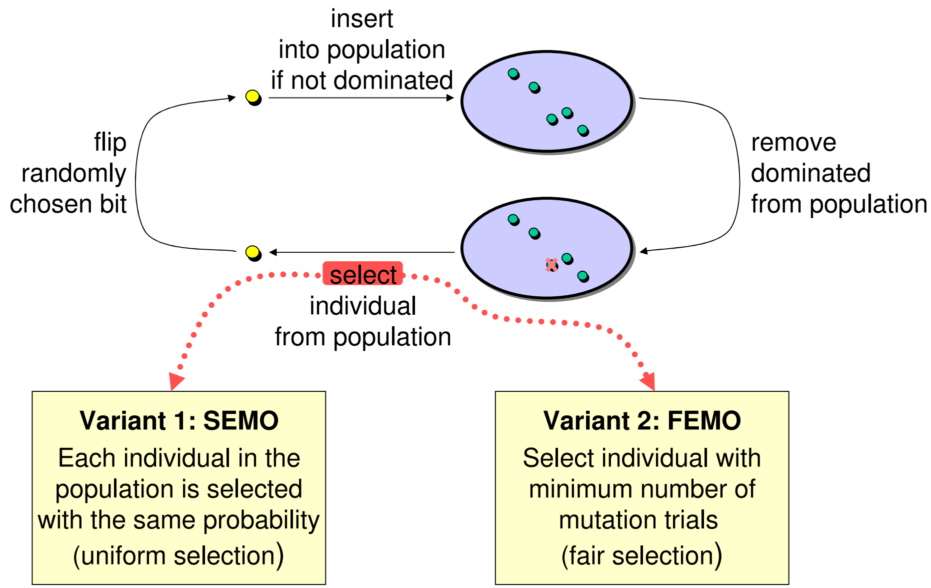
1	1	1	1	0	0	0

one bit per individual



[Laumanns et al. (2004)]

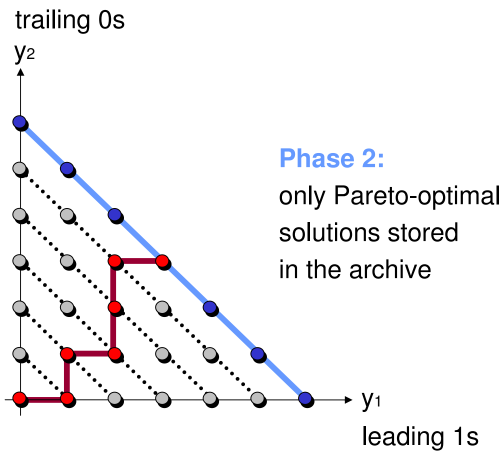
Two Simple Multiobjective EAs



The Good News

SEMO/FEMO behave like a single-objective EA until the Pareto set has been reached...

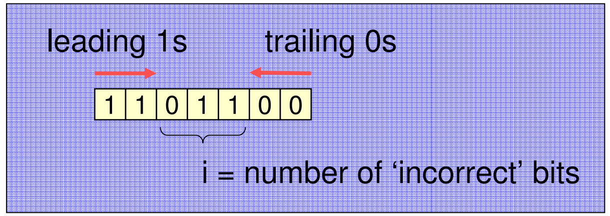
Phase 1:
only one solution stored in the archive



Phase 2:
only Pareto-optimal solutions stored in the archive

SEMO/FEMO: Sketch of the Analysis I

Phase 1: until first Pareto-optimal solution has been found

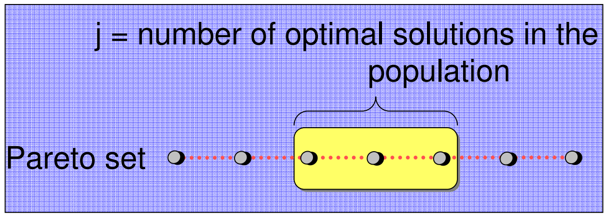


$i \rightarrow i\text{-}1$: probability of a successful mutation $\geq 1/n$
expected number of mutations = n

$i=n \rightarrow i=0$: at maximum n-1 steps (i=1 not possible)
expected overall number of mutations = $O(n^2)$

SEMO: Sketch of the Analysis II

Phase 2: from the first to all Pareto-optimal solutions

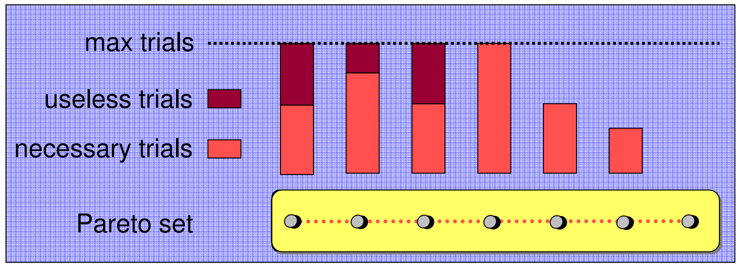


$j \rightarrow j+1$: probability of choosing an outer solution $\geq 1/j$, $\leq 2/j$
probability of a successful mutation $\geq 1/n$, $\leq 2/n$
expected number T_j of trials (mutations) $\geq nj/4$, $\leq nj$

$j=1 \rightarrow j=n$: at maximum n steps $\Rightarrow n^3/8 + n^2/8 \leq \sum T_j \leq n^3/2 + n^2/2$
expected overall number of mutations = $\Theta(n^3)$

FEMO: Sketch of the Analysis II

Phase 2: from the first to all Pareto-optimal solutions



Upper Bound:
'necessary' trials per solution $\leq 2n \log n$ with probability of at least $1 - O(1/n)$
'necessary' + 'useless' trials per solution $\leq 2n \log n$ with probability of at least $1 - O(1/n)$

FEMO: Results

Upper Bound:

overall number of mutation trials = $O(n^2 \log n)$ with probability $1 - O(1/n)$

Lower Bound:

overall number of mutations trials = $\Omega(n^2 \log n)$ with probability $1 - O(1/n)$

Expectation value:

expected number of mutation trials = $O(n^2 \log n)$

References

- D. Goldberg (1989): Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.
- M. Laumanns, L. Thiele, E. Zitzler (2004): Running Time Analysis of Multiobjective Evolutionary Algorithms on Pseudo-Boolean Functions. IEEE Transactions on Evolutionary Computation, IEEE Press, Vol. 8, No. 2, pp. 170-182.
- D. Wolpert, M. Macready (1997): No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, IEEE Press, Vol. 1, No. 1, pp. 67-82.