



Lecture

Computational Intelligence

Winter Term 2007/2008

Carsten Witt



Plans for Today

① Introduction

Summary of Last Lecture

② Typical Runs

Description of the Method

Application: Upper Bound

Application: Lower Bound

③ Complexity and The No Free Lunch Theorem

Introduction

The NFL Theorem, its Proof, and its Interpretation



Summary of Last Lecture

Drift Analysis

- is as a powerful tool for upper **and** lower bounds on the expected runtime.
- **We need:**
 - distance function
 - bounds on expected decrease in distance
 - expected initial distance (for lower bounds) or maximum distance (for upper bounds).
- **Application** to LEADINGONES: lower bound $\Omega(n^2)$ on the expected runtime of (1+1) EA as
 - expected initial distance $\geq n - 1$,
 - expected decrease in distance at most $2/n$,
 - mind the free-rider phenonemon.



Method of Typical Runs

Phase 1: Given EA starts with random initialization,

it reaches a population satisfying condition C_1 in at most $E(T_1)$ steps.

Phase 2: Given EA starts with a population satisfying condition C_1 ,

it reaches a population satisfying condition C_2 in at most $E(T_2)$ steps.

...

Phase k : Given EA starts with a population satisfying condition C_{k-1} ,

it reaches a population containing a global optimum in at most $E(T_k)$ steps.

Method of Typical Runs

Phase 1: Given EA starts with random initialization,

it reaches a population satisfying condition C_1 in at most $E(T_1)$ steps.

Phase 2: Given EA starts with a population satisfying condition C_1 ,

it reaches a population satisfying condition C_2 in at most $E(T_2)$ steps.

...

Phase k : Given EA starts with a population satisfying condition C_{k-1} ,

it reaches a population containing a global optimum in at most $E(T_k)$ steps.

This yields:
$$E(T_{EA,f}) \leq \sum_{i=1}^k E(T_i).$$



Method of Typical Runs

- Phase 1:** Given EA starts with random initialization, with probability at least $1 - p_1$, it reaches a population satisfying condition C_1 in at most T_1 steps.
- Phase 2:** Given EA starts with a population satisfying condition C_1 , with probability at least $1 - p_2$, it reaches a population satisfying condition C_2 in at most T_2 steps.
- ...
- Phase k :** Given EA starts with a population satisfying condition C_{k-1} , with probability at least $1 - p_k$, it reaches a population containing a global optimum in at most T_k steps.

This yields:
$$\text{Prob} \left(T_{\text{EA},f} \leq \sum_{i=1}^k T_i \right) \geq 1 - \sum_{i=1}^k p_i.$$



From Success Probability to Expected Optimization Time

What can be said about the expected optimization time if we only know $\text{Prob}\left(T_{\text{EA},f} \leq \sum_{i=1}^k T_i\right) \geq 1 - \sum_{i=1}^k p_i$?

Sometimes

“Phase 1: Given EA starts with random initialization”

can be replaced by

“Phase 1: EA may start with an arbitrary population”

From Success Probability to Expected Optimization Time

What can be said about the expected optimization time if we only know $\text{Prob}\left(T_{\text{EA},f} \leq \sum_{i=1}^k T_i\right) \geq 1 - \sum_{i=1}^k p_i$?

Sometimes

“Phase 1: Given EA starts with random initialization”

can be replaced by

“Phase 1: EA may start with an arbitrary population”

In this case, a **failure** in any phase can be described as a **restart**.

From Success Probability to Expected Optimization Time

What can be said about the expected optimization time if we only know $\text{Prob}\left(T_{\text{EA},f} \leq \sum_{i=1}^k T_i\right) \geq 1 - \sum_{i=1}^k p_i$?

Sometimes

“Phase 1: Given EA starts with random initialization”

can be replaced by

“Phase 1: EA may start with an arbitrary population”

In this case, a **failure** in any phase can be described as a **restart**.

This yields:
$$E(T_{\text{EA},f}) \leq \frac{\sum_{i=1}^k T_i}{1 - \sum_{i=1}^k p_i}.$$

A Concrete Example

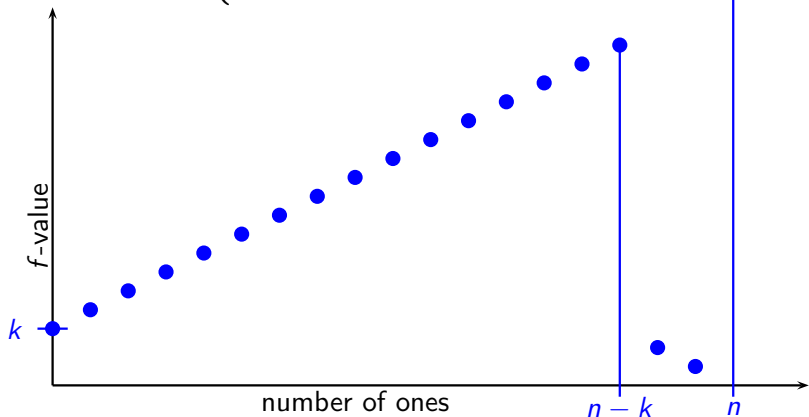
$\text{JUMP}_k(x): \{0, 1\}^n \rightarrow \mathbb{R}$ with $k \in \{1, 2, \dots, n\}$

$$\text{JUMP}_k(x) := \begin{cases} n - \text{ONEMAX}(x) & \text{if } n - k < \text{ONEMAX}(x) < n \\ k + \text{ONEMAX}(x) & \text{otherwise} \end{cases}$$

A Concrete Example

$\text{JUMP}_k(x): \{0, 1\}^n \rightarrow \mathbb{R}$ with $k \in \{1, 2, \dots, n\}$

$$\text{JUMP}_k(x) := \begin{cases} n - \text{ONEMAX}(x) & \text{if } n - k < \text{ONEMAX}(x) < n \\ k + \text{ONEMAX}(x) & \text{otherwise} \end{cases}$$



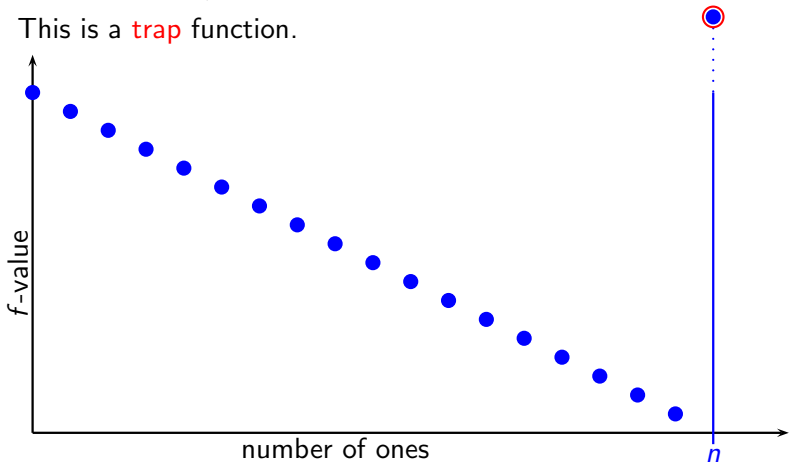
The Extreme Case: $k = n$

$$\text{JUMP}_n(x) := \begin{cases} n - \text{ONEMAX}(x) & \text{if } 0 \leq \text{ONEMAX}(x) < n \\ 2n & \text{otherwise} \end{cases}$$

The Extreme Case: $k = n$

$$\text{JUMP}_n(x) := \begin{cases} n - \text{ONEMAX}(x) & \text{if } 0 \leq \text{ONEMAX}(x) < n \\ 2n & \text{otherwise} \end{cases}$$

This is a **trap** function.





A Population-Based EA

A Population-Based EA

$(\mu+1)$ EA

1. Initialization

Choose $x_1, \dots, x_\mu \in \{0, 1\}^n$ uniformly at random.

2. Selection and Variation

Select z uniformly from x_1, \dots, x_μ .

$y := \text{standard-1}/n\text{-bit mutation}(z)$

3. Selection for Replacement

If $f(y) \geq \min\{f(x_1), \dots, f(x_\mu)\}$

Then replace some x_i with min. f -value by y .

4. “Stopping Criterion”

Continue at 2.

$(\mu+1)$ EA on JUMP_k

Theorem

Let $\mu = n^{O(1)}$. Then

- ① $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = O(n^k + \mu n \log n)$
- ② $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = \Omega(n^k)$

Tight bounds if $n^k \geq \mu n \log n$, e. g., for $\mu = \sqrt{n}$ choose $k = 2$.
 Parameter k tunes **difficulty** of the function.

$(\mu+1)$ EA on JUMP_k

Theorem

Let $\mu = n^{O(1)}$. Then

- ① $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = O(n^k + \mu n \log n)$
- ② $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = \Omega(n^k)$

Tight bounds if $n^k \geq \mu n \log n$, e. g., for $\mu = \sqrt{n}$ choose $k = 2$.
 Parameter k tunes **difficulty** of the function.

Method of proof: **typical runs** both for lower and upper bound



$(\mu+1)$ EA on JUMP_k

Theorem

Let $\mu = n^{O(1)}$. Then

- ① $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = O(n^k + \mu n \log n)$
- ② $E(T_{(\mu+1) \text{ EA, JUMP}_k}) = \Omega(n^k)$

Tight bounds if $n^k \geq \mu n \log n$, e. g., for $\mu = \sqrt{n}$ choose $k = 2$.
Parameter k tunes **difficulty** of the function.

Method of proof: **typical runs** both for lower and upper bound

Start with upper bound.

Upper Bound: Definition of the Phases

Notation:

$$\text{OPT} : \leftrightarrow n + k \in \{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\}$$

Upper Bound: Definition of the Phases

Notation:

$$\text{OPT} : \leftrightarrow n + k \in \{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\}$$

i	C_{i-1}	C_i	T_i
1	\emptyset	$\max\{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\} \geq n$	$O(\mu n \log n)$

Upper Bound: Definition of the Phases

Notation:

$$\text{OPT} : \leftrightarrow n + k \in \{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\}$$

i	C_{i-1}	C_i	T_i
1	\emptyset	$\max\{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\} \geq n$	$O(\mu n \log n)$
2	C_1	$\min\{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\} \geq n$	$O(\mu \log \mu)$

Upper Bound: Definition of the Phases

Notation:

$$\text{OPT} : \leftrightarrow n + k \in \{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\}$$

i	C_{i-1}	C_i	T_i
1	\emptyset	$\max\{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\} \geq n$	$O(\mu n \log n)$
2	C_1	$\min\{\text{JUMP}_k(x_1), \dots, \text{JUMP}_k(x_\mu)\} \geq n$	$O(\mu \log \mu)$
3	C_2	OPT	$O(n^k)$

Phase 1: Towards the Gap

Reaching some point x with $\text{JUMP}_k(x) \geq n$
is not more difficult than optimizing ONEMAX
(even if we start with more than $n - k$ one-bits).

Phase 1: Towards the Gap

Reaching some point x with $\text{JUMP}_k(x) \geq n$
 is not more difficult than optimizing ONEMAX
 (even if we start with more than $n - k$ one-bits).

For $\mu = 1$, $O(n \log n)$ follows.

Phase 1: Towards the Gap

Reaching some point x with $\text{JUMP}_k(x) \geq n$
 is not more difficult than optimizing ONEMAX
 (even if we start with more than $n - k$ one-bits).

For $\mu = 1$, $O(n \log n)$ follows.

For larger μ , observe:

With probability at least $\frac{1}{\mu}$, individual with highest fitness chosen.

Phase 1: Towards the Gap

Reaching some point x with $\text{JUMP}_k(x) \geq n$
 is not more difficult than optimizing ONEMAX
 (even if we start with more than $n - k$ one-bits).

For $\mu = 1$, $O(n \log n)$ follows.

For larger μ , observe:

With probability at least $\frac{1}{\mu}$, individual with highest fitness chosen.
 This implies $O(\mu n \log n)$ as expected length.

Phase 1: Towards the Gap

Reaching some point x with $\text{JUMP}_k(x) \geq n$
 is not more difficult than optimizing ONEMAX
 (even if we start with more than $n - k$ one-bits).

For $\mu = 1$, $O(n \log n)$ follows.

For larger μ , observe:

With probability at least $\frac{1}{\mu}$, individual with highest fitness chosen.
 This implies $O(\mu n \log n)$ as expected length.

If we were interested in the probability of a failure (here we aren't):

Markov's inequality: failure probability $p_1 \leq \varepsilon$
 for any constant $\varepsilon > 0$

Phase 2: At the Gap

Mind the standard mutation:

With probability $(1 - \frac{1}{n})^n \geq \frac{1}{2e}$, a clone of the parent is produced.

Say: individual is at the edge \leftrightarrow its JUMP-value is n .

Phase 2: At the Gap

Mind the standard mutation:

With probability $(1 - \frac{1}{n})^n \geq \frac{1}{2e}$, a clone of the parent is produced.

Say: individual is at the edge \leftrightarrow its JUMP-value is n .

Given k individuals at the edge (and no optimum reached):

- With probability k/μ , one of the edge individuals is chosen
- and with probability $\frac{1}{2e}$ this individual is cloned
- and a worse individual is deleted
- hence we have $k + 1$ edge individuals afterwards.

Phase 2: At the Gap

Mind the standard mutation:

With probability $(1 - \frac{1}{n})^n \geq \frac{1}{2e}$, a clone of the parent is produced.

Say: individual is at the edge \leftrightarrow its JUMP-value is n .

Given k individuals at the edge (and no optimum reached):

- With probability k/μ , one of the edge individuals is chosen
- and with probability $\frac{1}{2e}$ this individual is cloned
- and a worse individual is deleted
- hence we have $k + 1$ edge individuals afterwards.

Hence:

$E(\text{time to increase no. of edge individuals from } k \text{ to } k + 1) \leq \frac{2e\mu}{k}$



Phase 2: At the Gap

Mind the standard mutation:

With probability $(1 - \frac{1}{n})^n \geq \frac{1}{2e}$, a clone of the parent is produced.

Say: individual is at the edge \leftrightarrow its JUMP-value is n .

Given k individuals at the edge (and no optimum reached):

- With probability k/μ , one of the edge individuals is chosen
- and with probability $\frac{1}{2e}$ this individual is cloned
- and a worse individual is deleted
- hence we have $k + 1$ edge individuals afterwards.

Hence:

E(time to increase no. of edge individuals from k to $k + 1$) $\leq \frac{2e\mu}{k}$

Summing up: After expected time at most $\sum_{k=1}^{\mu-1} \frac{2e\mu}{k} = O(\mu \log \mu)$ we have only edge-individuals or are done anyway.

Phase 3: The Final Jump

Observation: Jump from the edge into the optimum has probability

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \geq \frac{1}{en^k}.$$



Phase 3: The Final Jump

Observation: Jump from the edge into the optimum has probability

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \geq \frac{1}{en^k}.$$

Recall: By assumption all individuals are at the edge.

Hence: Expected time for the final jump is $O(n^k)$.

Summing up:

$$\begin{aligned} E(T_{(\mu+1) \text{ EA, JUMP}_k}) &= O(\mu n \log n + \mu \log \mu + n^k) \\ &= O(\mu n \log n + n^k) \text{ since } \mu = n^{O(1)}. \quad \square \end{aligned}$$

Phase 3: The Final Jump

Observation: Jump from the edge into the optimum has probability

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \geq \frac{1}{en^k}.$$

Recall: By assumption all individuals are at the edge.

Hence: Expected time for the final jump is $O(n^k)$.

Summing up:

$$\begin{aligned} E(T_{(\mu+1) \text{ EA, JUMP}_k}) &= O(\mu n \log n + \mu \log \mu + n^k) \\ &= O(\mu n \log n + n^k) \text{ since } \mu = n^{O(1)}. \quad \square \end{aligned}$$

Question: What can be reused for the lower bound?

$(\mu+1)$ EA on JUMP_k : Lower Bound

Is the run considered for the upper bound really typical?

$(\mu+1)$ EA on JUMP_k : Lower Bound

Is the run considered for the upper bound really typical?

Yes, we will show an exponentially small failure probability.

After Phase 2 the probability of finding OPT is $O(n^{-k})$,
and then the expected time to find OPT is $\Omega(n^k)$.

Only problem: Individual might jump into optimum
before Phase 2 is over (call such jump a “failure”).

$(\mu+1)$ EA on JUMP_k : Lower Bound

Is the run considered for the upper bound really typical?

Yes, we will show an exponentially small failure probability.

After Phase 2 the probability of finding OPT is $O(n^{-k})$,
and then the expected time to find OPT is $\Omega(n^k)$.

Only problem: Individual might jump into optimum
before Phase 2 is over (call such jump a “failure”).

No problem if this individual is left of the edge
since even more than k bits would have to flip.

If k is large, we might however start “in the valley”,
i. e., with more than $n - k$ ones.

Lower Bound: Bounding the Failure Probability (1/2)

Consider individual in the valley with maximal number of ones.

Notation: Call this maximal number h .

Observation: h cannot increase in the run unless optimum reached.

Lower Bound: Bounding the Failure Probability (1/2)

Consider individual in the valley with maximal number of ones.

Notation: Call this maximal number h .

Observation: h cannot increase in the run unless optimum reached.

Therefore bound h for the initial population

Chernoff bound:

$$\text{Prob}(\text{individual initialized} \geq 2n/3 \text{ ones}) \leq e^{-n/36}$$

Union bound:

$$\text{Prob}(\exists \text{ initially individual with} \geq 2n/3 \text{ ones}) \leq \mu e^{-n/36} = e^{-\Omega(n)}.$$

Thus with probability $1 - e^{-\Omega(n)}$ we have $h \leq \frac{2n}{3}$,
then at least $n/3$ bits have to flip to jump into the optimum.

Lower Bound: Bounding the Failure Probability (2/2)

We have: With probability $1 - e^{-\Omega(n)}$ an event occurs such that jumping into the optimum **at any time** has prob. $\leq n^{-n/3}$.

Altogether at any time failure prob. $\leq e^{-\Omega(n)} + n^{-n/3} = e^{-\Omega(n)}$.

Lower Bound: Bounding the Failure Probability (2/2)

We have: With probability $1 - e^{-\Omega(n)}$ an event occurs such that jumping into the optimum **at any time** has prob. $\leq n^{-n/3}$.

Altogether at any time failure prob. $\leq e^{-\Omega(n)} + n^{-n/3} = e^{-\Omega(n)}$.

Recall: We considered $h > n - k$, yet at the end of Phase 2, the probability of finding the optimum is $O(n^{-k})$.

Need probability of a failure before Phase 2 ends.

Notation: $T =$ random length of Phase 2

Lower Bound: Bounding the Failure Probability (2/2)

We have: With probability $1 - e^{-\Omega(n)}$ an event occurs such that jumping into the optimum **at any time** has prob. $\leq n^{-n/3}$.

Altogether at any time failure prob. $\leq e^{-\Omega(n)} + n^{-n/3} = e^{-\Omega(n)}$.

Recall: We considered $h > n - k$, yet at the end of Phase 2, the probability of finding the optimum is $O(n^{-k})$.

Need probability of a failure before Phase 2 ends.

Notation: $T =$ random length of Phase 2

Then: As probability of failure within t steps is $\leq t \cdot e^{-\Omega(n)}$,
 Prob(failure before the end of Phase 2)
 $\leq \sum_{t=0}^{\infty} \text{Prob}(T = t) \cdot (t \cdot e^{-\Omega(n)}) = E(T) \cdot e^{-\Omega(n)}$
 $= O(\mu n \log n) e^{-\Omega(n)} = e^{-\Omega(n)}$ as $\mu = n^{O(1)}$.

Altogether: expected runtime $(1 - e^{-\Omega(n)})\Omega(n^k) = \Omega(n^k)$. \square

On the Way to More General Results

Observation: Analysis of $E(T_{A,f})$ is **difficult** and **does not help much** if I go to EA A' and fitness f' even if A' and f' are similar to A and f .

Can we not obtain more general results?

Alternatively, can we find out which evolutionary algorithm is the best?

Is there a single best evolutionary algorithm?

On the Way to More General Results

Observation: Analysis of $E(T_{A,f})$ is **difficult** and **does not help much** if I go to EA A' and fitness f' even if A' and f' are similar to A and f .

Can we not obtain more general results?

Alternatively, can we find out which evolutionary algorithm is the best?

Is there a single best evolutionary algorithm?

They say: “There ain’t no such thing as a free lunch.”

NFL Theorem: “On average, all randomized search heuristics perform equal.”



On the Way to More General Results

Observation: Analysis of $E(T_{A,f})$ is **difficult** and **does not help much** if I go to EA A' and fitness f' even if A' and f' are similar to A and f .

Can we not obtain more general results?

Alternatively, can we find out which evolutionary algorithm is the best?

Is there a single best evolutionary algorithm?

They say: “There ain’t no such thing as a free lunch.”

NFL Theorem: “On average, all randomized search heuristics perform equal.”

Is this surprising?

What the NFL Theorem is about

Basic Notions:

- (optimization) problem, e. g., TSP
- problem instance, e. g., cities with distances

What the NFL Theorem is about

Basic Notions:

- (optimization) problem, e. g., TSP
- problem instance, e. g., cities with distances

Usually: problem and problem instance known to the optimization algorithm

What the NFL Theorem is about

Basic Notions:

- (optimization) problem, e. g., TSP
- problem instance, e. g., cities with distances

Usually: problem and problem instance known to the optimization algorithm

Black Box Optimization: problem known,
problem instance unknown

More formally:

Optimize some **unknown** $f: S \rightarrow W$ ($f \in \mathcal{F}$, \mathcal{F} known).

An oracle presents for any input $x \in S$ the output $f(x)$.

We call one access of the oracle a **function evaluation**.

The NFL Theorem deals with Black Box Optimization in this sense.

The NFL Theorem

The NFL Theorem measures performance by means of function evaluations.

The NFL Theorem

The NFL Theorem measures performance by means of function evaluations.

The NFL Theorem makes a statement for all general randomized search heuristics working on some function $f: S \rightarrow W$ where S and W are finite.

The NFL Theorem

The NFL Theorem measures performance by means of function evaluations.

The NFL Theorem makes a statement for all general randomized search heuristics working on some function $f: S \rightarrow W$ where S and W are finite.

Is this a relevant restriction?

Obviously, if S is finite, having W finite is no restriction.

Having S finite seems to be a restriction.

Real algorithms on real computers can only deal with finite S .

Making Things Easier...

Since S and W are finite,
there is only a finite number of different **deterministic** algorithms.

Making Things Easier...

Since S and W are finite,
there is only a finite number of different **deterministic** algorithms.

In such a scenario,
any randomized algorithm can be described as
probability distribution over deterministic algorithms.

Making Things Easier...

Since S and W are finite,
there is only a finite number of different **deterministic** algorithms.

In such a scenario,
any randomized algorithm can be described as
probability distribution over deterministic algorithms.

Similar to “guess and verify” Turing machines:
Make all random choices first, then compute deterministically.

Conclusion: We can concentrate on deterministic algorithms.

Describing Deterministic Algorithms

Algorithm corresponds to tree with

- nodes labelled with search points $x \in S$,
- root labelled with first search point $s_1 \in S$,
- edges labelled with function values $f(x)$.

Observation: It is easy to avoid sampling points more than once.

Describing Deterministic Algorithms

Algorithm corresponds to tree with

- nodes labelled with search points $x \in S$,
- root labelled with first search point $s_1 \in S$,
- edges labelled with function values $f(x)$.

Observation: It is easy to avoid sampling points more than once.

Conclusion: We can assume that

- any point in S is sampled at most once,
- the trees have depth at most $|S|$.

Describing Deterministic Algorithms

Algorithm corresponds to tree with

- nodes labelled with search points $x \in S$,
- root labelled with first search point $s_1 \in S$,
- edges labelled with function values $f(x)$.

Observation: It is easy to avoid sampling points more than once.

Conclusion: We can assume that

- any point in S is sampled at most once,
- the trees have depth at most $|S|$.

Algorithm A on function f :

$$T_A(f, t) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_t, f(s_t)) \rangle$$

even shorter:

$$V_A(f, t) = \langle f(s_1), f(s_2), \dots, f(s_t) \rangle$$

Measuring Performance

We measure performance by means of different function evaluations.

Performance measure: $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

Measuring Performance

We measure performance by means of different function evaluations.

Performance measure: $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

Example:

$$M(V_A(f, |S|)) = \min \{i \mid f(s_i) = \max\{f(x) \mid x \in S\}\}$$

Measuring Performance

We measure performance by means of different function evaluations.

Performance measure: $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

Example:

$$M(V_A(f, |S|)) = \min \{i \mid f(s_i) = \max\{f(x) \mid x \in S\}\}$$

Without loss of generality, we assume that each algorithm visits each $x \in S$ at some point of time.

The NFL Theorem

Theorem

S, W finite sets, $\mathcal{F} \subseteq \{f: S \rightarrow W\}$.

On average over all $f \in \mathcal{F}$,

the number of different function evaluations

an algorithm makes for any performance measure M

is equal for all algorithms

if and only if \mathcal{F} is closed under permutations of the search space.

The NFL Theorem

Theorem

S, W finite sets, $\mathcal{F} \subseteq \{f: S \rightarrow W\}$.

On average over all $f \in \mathcal{F}$,

the number of different function evaluations

an algorithm makes for any performance measure M

is equal for all algorithms

if and only if \mathcal{F} is closed under permutations of the search space.

Permutations of the search space:

$f: S \rightarrow W, \sigma \in \text{Perm}(S)$:

$\sigma f: S \rightarrow W$ is given by $\sigma f(x) := f(\sigma^{-1}(x))$

The NFL Theorem

Theorem

S, W finite sets, $\mathcal{F} \subseteq \{f: S \rightarrow W\}$.

On average over all $f \in \mathcal{F}$,

the number of different function evaluations

an algorithm makes for any performance measure M

is equal for all algorithms

if and only if \mathcal{F} is closed under permutations of the search space.

Permutations of the search space:

$f: S \rightarrow W, \sigma \in \text{Perm}(S)$:

$\sigma f: S \rightarrow W$ is given by $\sigma f(x) := f(\sigma^{-1}(x))$

$\mathcal{F} \subseteq \{f: S \rightarrow W\}$ is called

closed under permutations of the search space

if and only if

$\forall f \in \mathcal{F}, \sigma \in \text{Perm}(S): \sigma f \in \mathcal{F}$.

Proof: “ \mathcal{F} c. u. p. \Rightarrow NFL”

It suffices to consider deterministic algorithms.

Proof: “ \mathcal{F} c. u. p. \Rightarrow NFL”

It suffices to consider deterministic algorithms.

Let A, A' be two deterministic algorithms, $f \in \mathcal{F}$.
Consider $V_A(f, |S|) = \langle f(s_1), f(s_2), \dots, f(s_{|S|}) \rangle$.

We define $\sigma_f \in \text{Perm}(S)$:

Proof: “ \mathcal{F} c. u. p. \Rightarrow NFL”

It suffices to consider deterministic algorithms.

Let A, A' be two deterministic algorithms, $f \in \mathcal{F}$.
Consider $V_A(f, |S|) = \langle f(s_1), f(s_2), \dots, f(s_{|S|}) \rangle$.

We define $\sigma_f \in \text{Perm}(S)$:

Let s'_1 be the root's label in A' .
 $\sigma_f(s_1) := s'_1$

For $i \in \{1, 2, \dots, |S| - 1\}$,
let s'_{i+1} be the $f(s_i)$ -son of $\sigma_f(s_i)$ in A' and $\sigma_f(s_{i+1}) := s'_{i+1}$

Obviously: $V_A(f, |S|) = V_{A'}(\sigma_f f, |S|)$.

Proof: “ \mathcal{F} c. u. p. \Rightarrow NFL” (continued)

$$\mathcal{F} \text{ c. u. p.} \Rightarrow \sigma_f f \in \mathcal{F}$$

Observation: $\{\sigma_f f \mid f \in \mathcal{F}\} = \mathcal{F}$

Proof: “ \mathcal{F} c. u. p. \Rightarrow NFL” (continued)

$$\mathcal{F} \text{ c. u. p. } \Rightarrow \sigma_f f \in \mathcal{F}$$

Observation: $\{\sigma_f f \mid f \in \mathcal{F}\} = \mathcal{F}$

Conclusion:

$$\sum_{f \in \mathcal{F}} M(V_A(f, |S|)) = \sum_{f \in \mathcal{F}} M(V_{A'}(f, |S|))$$

A and A' perform equal on average over \mathcal{F} .

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL”

Let \mathcal{F} be not c. u. p.

There exist $f \in \mathcal{F}$ and $\sigma \in \text{Perm}(S)$ with $\sigma f \notin \mathcal{F}$.

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL”

Let \mathcal{F} be not c. u. p.

There exist $f \in \mathcal{F}$ and $\sigma \in \text{Perm}(S)$ with $\sigma f \notin \mathcal{F}$.

Let A be an algorithm. Consider $V_A(f, |S|)$.

Define $M(V) := \begin{cases} 1 & \text{if } V = V_A(f, |S|) \\ 0 & \text{otherwise} \end{cases}$

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL”

Let \mathcal{F} be not c. u. p.

There exist $f \in \mathcal{F}$ and $\sigma \in \text{Perm}(S)$ with $\sigma f \notin \mathcal{F}$.

Let A be an algorithm. Consider $V_A(f, |S|)$.

Define $M(V) := \begin{cases} 1 & \text{if } V = V_A(f, |S|) \\ 0 & \text{otherwise} \end{cases}$

Obviously:
$$\frac{\sum_{g \in \mathcal{F}} M(V_A(g, |S|))}{|\mathcal{F}|} = \frac{1}{|\mathcal{F}|}$$

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL” (continued)

Consider $T_A(f, |S|) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

Define algorithm B : Sample $\sigma(s_1), \sigma(s_2), \dots, \sigma(s_{|S|})$.

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL” (continued)

Consider $T_A(f, |S|) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

Define algorithm B : Sample $\sigma(s_1), \sigma(s_2), \dots, \sigma(s_{|S|})$.

Observation: $M(V_B(g, |S|)) = 1 \Leftrightarrow g = \sigma f$

Proof: “ \mathcal{F} not c. u. p. \Rightarrow no NFL” (continued)

Consider $T_A(f, |S|) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

Define algorithm B : Sample $\sigma(s_1), \sigma(s_2), \dots, \sigma(s_{|S|})$.

Observation: $M(V_B(g, |S|)) = 1 \Leftrightarrow g = \sigma f$

Conclusion: $\frac{\sum_{g \in \mathcal{F}} M(V_B(g, |S|))}{|\mathcal{F}|} = 0 \neq \frac{1}{|\mathcal{F}|} = \frac{\sum_{g \in \mathcal{F}} M(V_A(g, |S|))}{|\mathcal{F}|}$

□

What does NFL mean?

Is “closed under permutations” a typical property?

Theorem

Only a fraction of

$$\frac{2^{\binom{|S|+|W|-1}{|S|}} - 1}{2^{|W||S|} - 1}$$

of all non-empty function classes $\mathcal{F} \subseteq W^S$ is c. u. p.

(without proof)

What does NFL mean?

Is “closed under permutations” a typical property?

Theorem

Only a fraction of

$$\frac{2^{\binom{|S|+|W|-1}{|S|}} - 1}{2^{|W|^{|S|}} - 1}$$

of all non-empty function classes $\mathcal{F} \subseteq W^S$ is c. u. p.

(without proof)

Conclusion: Random function classes are almost surely not c. u. p.

Are “Practical” Function Classes c. u. p.?

Consider non-trivial neighborhood on S :

$N: S \times S \rightarrow \{0, 1\}$ with $\exists s_1, s_2, s_3, s_4: N(s_1, s_2) = 1, N(s_3, s_4) = 0$.

Are “Practical” Function Classes c. u. p.?

Consider non-trivial neighborhood on S :

$N: S \times S \rightarrow \{0, 1\}$ with $\exists s_1, s_2, s_3, s_4: N(s_1, s_2) = 1, N(s_3, s_4) = 0$.

“Practical” function classes have some property defined on N :

- some degree of smoothness
- number of local optima not maximal
- local maxima and local minima not neighbors
- ...

Are “Practical” Function Classes c. u. p.?

Consider non-trivial neighborhood on S :

$N: S \times S \rightarrow \{0, 1\}$ with $\exists s_1, s_2, s_3, s_4: N(s_1, s_2) = 1, N(s_3, s_4) = 0$.

“Practical” function classes have some property defined on N :

- some degree of smoothness
- number of local optima not maximal
- local maxima and local minima not neighbors
- ...

Main observation: Neighborhood is not invariant under permutations.



Are “Practical” Function Classes c. u. p.?

Consider non-trivial neighborhood on S :

$N: S \times S \rightarrow \{0, 1\}$ with $\exists s_1, s_2, s_3, s_4: N(s_1, s_2) = 1, N(s_3, s_4) = 0$.

“Practical” function classes have some property defined on N :

- some degree of smoothness
- number of local optima not maximal
- local maxima and local minima not neighbors
- ...

Main observation: Neighborhood is not invariant under permutations.

“Practical” function classes are **not** c. u. p.

Criticism

In practice, the function class \mathcal{F} has **restricted complexity**.

Criticism

In practice, the function class \mathcal{F} has **restricted complexity**.

Complexity:

- limited computation time for $f(x)$
- limited description space (Kolmogorov complexity)
- limited circuit size
- ...

Criticism

In practice, the function class \mathcal{F} has **restricted complexity**.

Complexity:

- limited computation time for $f(x)$
- limited description space (Kolmogorov complexity)
- limited circuit size
- ...

How much can be gained by taking just this into account?

The ANFL Theorem

Theorem

Let $S := \{0, 1\}^n$, $W := \{0, 1, \dots, N - 1\}$, $f: S \rightarrow W$,

A a randomized black box algorithm for any $\mathcal{F} \subseteq W^S$.

There are at least $N^{2^{n/3}-1}$ functions $f': S \rightarrow W \cup \{N\}$, such that A fails to optimize f' within $2^{n/3}$ function evaluations with probability at least $1 - 2^{-n/3}$.

Exponentially many of these functions have a complexity (computation time, Kolmogorov complexity, circuit size) that is only $O(n)$ larger than that of f .

(without proof)

What does the ANFL mean?

- Black box algorithms performing well on some function must be much worse on many other similar functions.
- Restricting the complexity of \mathcal{F} is insufficient to guarantee good performance.
- If not much is known about \mathcal{F} , evolutionary algorithms may fail.