

# CS4618 Artificial Intelligence I

Today: Collecting Coupons  
Assessment of  
Randomised Search Heuristics

Thomas Jansen

October 31<sup>st</sup>

# Plans for Today

- ① Coupon Collector's Theorem
  - Introduction
  - Result and Proof
- ② Assessing Randomised Search Heuristics
  - Motivation
  - No Free Lunch
- ③ Summary
  - Summary & Take Home Message

# Collecting Coupons

Consider Collecting coupons,  $n$  different types of coupons

# Collecting Coupons

**Consider** Collecting coupons,  $n$  different types of coupons

**Assumption**  $\forall i \in \{1, 2, \dots, n\}$ :

each time, independently,  $\text{Prob}(\text{get coupon type } i) = \frac{1}{n}$

# Collecting Coupons

**Consider** Collecting coupons,  $n$  different types of coupons

**Assumption**  $\forall i \in \{1, 2, \dots, n\}$ :

each time, independently,  $\text{Prob}(\text{get coupon type } i) = \frac{1}{n}$

How many coupons do we need until we have a complete collection?

# Collecting Coupons

**Consider** Collecting coupons,  $n$  different types of coupons

**Assumption**  $\forall i \in \{1, 2, \dots, n\}$ :

each time, independently,  $\text{Prob}(\text{get coupon type } i) = \frac{1}{n}$

How many coupons do we need until we have a complete collection?

**Observation** this number is random variable  $T$

# Collecting Coupons

**Consider** Collecting coupons,  $n$  different types of coupons

**Assumption**  $\forall i \in \{1, 2, \dots, n\}$ :

each time, independently,  $\text{Prob}(\text{get coupon type } i) = \frac{1}{n}$

How many coupons do we need until we have a complete collection?

**Observation** this number is random variable  $T$

**Questions** What is  $E(T)$ ?

How likely are deviations from  $E(T)$ ?

## A Detour: Landau Notation

**Observation**    being less exact can simplify things a lot

## A Detour: Landau Notation

**Observation** being less exact can simplify things a lot

**Idea** Characterise functions according to their “most important term”

## A Detour: Landau Notation

**Observation** being less exact can simplify things a lot

**Idea** Characterise functions according to their “most important term”

**Different point of view** Characterise functions according to their “order of growth”

## A Detour: Landau Notation

**Observation** being less exact can simplify things a lot

**Idea** Characterise functions according to their “most important term”

**Different point of view** Characterise functions according to their “order of growth”

**Remember** for comparing numbers  $\leq, \geq, =, <, >$

## A Detour: Landau Notation

**Observation** being less exact can simplify things a lot

**Idea** Characterise functions according to their “most important term”

**Different point of view** Characterise functions according to their “order of growth”

**Remember** for comparing numbers  $\leq, \geq, =, <, >$

**Define** comparisons for functions similarly

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \iff \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$  “ $\leq$ ”

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$   
 $f$  and  $g$  grow equally fast    “ $=$ ”

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$   
 $f$  and  $g$  grow equally fast    “ $=$ ”
- $f = o(g) \quad :\Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$   
 $f$  and  $g$  grow equally fast    “ $=$ ”
- $f = o(g) \quad :\Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$   
 $f$  grows slower than  $g$     “ $<$ ”

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$   
 $f$  and  $g$  grow equally fast    “ $=$ ”
- $f = o(g) \quad :\Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$   
 $f$  grows slower than  $g$     “ $<$ ”
- $f = \omega(g) \quad :\Leftrightarrow g = o(f)$

# Landau Notation

## Definition

For  $f, g: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  define

- $f = O(g) \quad :\Leftrightarrow \exists n_0, c > 0: \forall n \geq n_0: f(n) \leq c \cdot g(n)$   
 $f$  grows not faster than  $g$     “ $\leq$ ”
- $f = \Omega(g) \quad :\Leftrightarrow g = O(f)$   
 $f$  grows not slower than  $g$     “ $\geq$ ”
- $f = \Theta(g) \quad :\Leftrightarrow f = O(g) \wedge f = \Omega(g)$   
 $f$  and  $g$  grow equally fast    “ $=$ ”
- $f = o(g) \quad :\Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$   
 $f$  grows slower than  $g$     “ $<$ ”
- $f = \omega(g) \quad :\Leftrightarrow g = o(f)$   
 $f$  grows faster than  $g$     “ $>$ ”

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

*Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.*

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

*Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.*

①  $E(T) = n \ln n + O(n)$

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

*Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.*

- 1  $E(T) = n \ln n + O(n)$
- 2  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

*Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.*

- 1  $E(T) = n \ln n + O(n)$
- 2  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$
- 3  $\forall c \in \mathbb{R}: \text{Prob}(T > n \ln n + cn) = 1 - e^{-e^{-c}}$

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.

- ①  $E(T) = n \ln n + O(n)$
- ②  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$
- ③  $\forall c \in \mathbb{R}: \text{Prob}(T > n \ln n + cn) = 1 - e^{-e^{-c}}$

## Proof

**Partition** random process into **disjoint phases**  
 in phase  $i$ , exactly  $i$  different types have been obtained  
**length** of phase  $i$  is  $T_i$

# The Coupon Collector's Problem

## Theorem (Coupon Collector's Theorem)

Let  $T$  denote the number of coupons obtained until all  $n$  types of coupons are present for the first time.

- ①  $E(T) = n \ln n + O(n)$
- ②  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$
- ③  $\forall c \in \mathbb{R}: \text{Prob}(T > n \ln n + cn) = 1 - e^{-e^{-c}}$

## Proof

**Partition** random process into **disjoint phases**  
 in phase  $i$ , exactly  $i$  different types have been obtained  
**length** of phase  $i$  is  $T_i$

**Observation**  $T = \sum_{i=0}^{n-1} T_i \Rightarrow E(T) = \sum_{i=0}^{n-1} E(T_i)$



# Phases in Obtaining Coupons

# Phases in Obtaining Coupons

in phase  $i$     Prob (obtain  $(i + 1)$ -th type) =  $(n - i)/n$

# Phases in Obtaining Coupons

in phase  $i$     Prob (obtain  $(i + 1)$ -th type) =  $(n - i)/n$

thus     $E(T_i) = \frac{n}{n-i}$

## Phases in Obtaining Coupons

in phase  $i$     Prob (obtain  $(i + 1)$ -th type) =  $(n - i)/n$

thus     $E(T_i) = \frac{n}{n-i}$

$$\begin{aligned} \Rightarrow E(T) &= \sum_{i=0}^{n-1} \frac{n}{n-i} \\ &= n \sum_{i=1}^n \frac{1}{i} = n \ln n + O(n) \quad \checkmark \end{aligned}$$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

Define event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

Define event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

Observation  $\text{Prob}(S_{i,t}) =$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

**Define** event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

**Observation**  $\text{Prob}(S_{i,t}) = \left(1 - \frac{1}{n}\right)^t \leq e^{-t/n}$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

**Define** event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

**Observation**  $\text{Prob}(S_{i,t}) = \left(1 - \frac{1}{n}\right)^t \leq e^{-t/n}$

**Observation**  $\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

Define event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

Observation  $\text{Prob}(S_{i,t}) = \left(1 - \frac{1}{n}\right)^t \leq e^{-t/n}$

Observation  $\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$

estimate  $\text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right) \leq \sum_{i=1}^n \text{Prob}(S_{i,t}) \leq n \cdot e^{-t/n}$

## Proof of the 2nd Statement

need to prove  $\forall \beta > 1: \text{Prob}(T > \beta n \ln n) \leq n^{-(\beta-1)}$

Define event  $S_{i,t}$   
type  $i$  not obtained within first  $t$  coupons

Observation  $\text{Prob}(S_{i,t}) = \left(1 - \frac{1}{n}\right)^t \leq e^{-t/n}$

Observation  $\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$

estimate  $\text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right) \leq \sum_{i=1}^n \text{Prob}(S_{i,t}) \leq n \cdot e^{-t/n}$

choose  $t = \beta n \ln n$

$\text{Prob}(T > \beta n \ln n) \leq n \cdot e^{-\beta \ln n} = n^{-(\beta-1)}$  ✓

## Proof of the 3rd Statement

need to prove  $\forall c \in \mathbb{R}: \text{Prob}(T > n \ln n + cn) = 1 - e^{-e^{-c}}$

we know  $\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$

## Proof of the 3rd Statement

need to prove  $\forall c \in \mathbb{R}: \text{Prob}(T > n \ln n + cn) = 1 - e^{-e^{-c}}$

we know  $\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$

principle of inclusion and exclusion

$$\begin{aligned} \text{Prob}\left(\bigcup_{i=1}^n A_i\right) &= \sum_{i=1}^n \text{Prob}(A_i) - \sum_{1 \leq i < j \leq n} \text{Prob}(A_i \cap A_j) \\ &+ \sum_{1 \leq i < j < k \leq n} \text{Prob}(A_i \cap A_j \cap A_k) + \dots \\ &+ (-1)^{n+1} \sum_{1 \leq h_1 < h_2 < \dots < h_n \leq n} \text{Prob}\left(\bigcap_{j=1}^n A_{h_j}\right) \end{aligned}$$

## Boole-Bonferroni Inequalities

for even  $k$   $\text{Prob} \left( \bigcup_{i=1}^n A_i \right)$

$$\geq \sum_{i=1}^k (-1)^{i+1} \sum_{1 \leq h_1 < h_2 < \dots < h_i \leq n} \text{Prob} \left( \bigcap_{j=1}^i A_{h_j} \right)$$

for odd  $k$   $\text{Prob} \left( \bigcup_{i=1}^n A_i \right)$

$$\leq \sum_{i=1}^k (-1)^{i+1} \sum_{1 \leq h_1 < h_2 < \dots < h_i \leq n} \text{Prob} \left( \bigcap_{j=1}^i A_{h_j} \right)$$

## Boole-Bonferroni Inequalities

for even  $k$   $\text{Prob} \left( \bigcup_{i=1}^n A_i \right)$

$$\geq \sum_{i=1}^k (-1)^{i+1} \sum_{1 \leq h_1 < h_2 < \dots < h_i \leq n} \text{Prob} \left( \bigcap_{j=1}^i A_{h_j} \right)$$

for odd  $k$   $\text{Prob} \left( \bigcup_{i=1}^n A_i \right)$

$$\leq \sum_{i=1}^k (-1)^{i+1} \sum_{1 \leq h_1 < h_2 < \dots < h_i \leq n} \text{Prob} \left( \bigcap_{j=1}^i A_{h_j} \right)$$

Notation  $P_l = \sum_{1 \leq h_1 < h_2 < \dots < h_l \leq n} \text{Prob} \left( \bigcap_{j=1}^l A_{h_j} \right)$

# Application of Boole-Bonferroni Inequalities

$$\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$$

# Application of Boole-Bonferroni Inequalities

$$\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$$

$$\text{Prob}(T > n \ln n + cn) = \sum_{i=1}^n (-1)^{i+1} P_i$$

## Application of Boole-Bonferroni Inequalities

$$\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$$

$$\text{Prob}(T > n \ln n + cn) = \sum_{i=1}^n (-1)^{i+1} P_i$$

$$n \in \{2k, 2k + 1\}$$

$$\sum_{i=1}^{2k} (-1)^{i+1} P_i \leq \text{Prob}(T > n \ln n + cn) \leq \sum_{i=1}^{2k+1} (-1)^{i+1} P_i$$

## Application of Boole-Bonferroni Inequalities

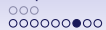
$$\forall t: \text{Prob}(T > t) = \text{Prob}\left(\bigcup_{i=1}^n S_{i,t}\right)$$

$$\text{Prob}(T > n \ln n + cn) = \sum_{i=1}^n (-1)^{i+1} P_i$$

$$n \in \{2k, 2k + 1\}$$

$$\sum_{i=1}^{2k} (-1)^{i+1} P_i \leq \text{Prob}(T > n \ln n + cn) \leq \sum_{i=1}^{2k+1} (-1)^{i+1} P_i$$

from structure of problem      estimating  $P_i$



# Estimating $P_i$

Estimating  $P_l$ 

$$P_l = \sum_{1 \leq h_1 < h_2 < \dots < h_l \leq n} \text{Prob} \left( \bigcap_{j=1}^l A_{h_j} \right)$$

coupons homogeneous

$$= \binom{n}{l} \text{Prob} \left( \bigcap_{j=1}^l A_j \right)$$

Estimating  $P_i$ 

$$P_l = \sum_{1 \leq h_1 < h_2 < \dots < h_l \leq n} \text{Prob} \left( \bigcap_{j=1}^l A_{h_j} \right)$$

coupons homogeneous  $\stackrel{=}{=}$   $\binom{n}{l} \text{Prob} \left( \bigcap_{j=1}^l A_j \right)$

structure of problem  $A_j = S_{j,n \ln n + cn}$

Estimating  $P_l$ 

$$P_l = \sum_{1 \leq h_1 < h_2 < \dots < h_l \leq n} \text{Prob} \left( \bigcap_{j=1}^l A_{h_j} \right)$$

coupons homogeneous  $\stackrel{=}{=} \binom{n}{l} \text{Prob} \left( \bigcap_{j=1}^l A_j \right)$

structure of problem  $A_j = S_{j, n \ln n + cn}$

Observation  $\text{Prob} \left( \bigcap_{j=1}^l A_j \right) = \left( 1 - \frac{l}{n} \right)^{n \ln n + cn}$

Estimating  $P_l$ 

$$P_l = \sum_{1 \leq h_1 < h_2 < \dots < h_l \leq n} \text{Prob} \left( \bigcap_{j=1}^l A_{h_j} \right)$$

coupons homogeneous  $\stackrel{=}{=} \binom{n}{l} \text{Prob} \left( \bigcap_{j=1}^l A_j \right)$

structure of problem  $A_j = S_{j, n \ln n + cn}$

Observation  $\text{Prob} \left( \bigcap_{j=1}^l A_j \right) = \left( 1 - \frac{l}{n} \right)^{n \ln n + cn}$

thus  $P_l = \binom{n}{l} \left( 1 - \frac{l}{n} \right)^{n \ln n + cn}$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k (n-k)!}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k (n-k)!} = \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k(n-k)!} = \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n}$$

thus

$$\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k (n-k)!} = \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n}$$

thus

$$\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} = \lim_{n \rightarrow \infty} n^k \left(1 - \frac{k}{n}\right)^{(n/k)k(\ln n + c)}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k(n-k)!} = \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n}$$

thus

$$\begin{aligned} \lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} &= \lim_{n \rightarrow \infty} n^k \left(1 - \frac{k}{n}\right)^{(n/k)k(\ln n + c)} \\ &= \lim_{n \rightarrow \infty} n^k e^{-k \ln n - ck} \end{aligned}$$

## Detour: Calculation

$$k! \cdot \frac{1}{n^k} \cdot \binom{n}{k} = \frac{n!}{n^k(n-k)!} = \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n}$$

thus

$$\begin{aligned} \lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} &= \lim_{n \rightarrow \infty} n^k \left(1 - \frac{k}{n}\right)^{(n/k)k(\ln n + c)} \\ &= \lim_{n \rightarrow \infty} n^k e^{-k \ln n - ck} = e^{-ck} \end{aligned}$$

## Completing the Proof

We know  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn)$

$$= \lim_{n \rightarrow \infty} \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} \left(1 - \frac{i}{n}\right)^{n \ln n + cn}$$

and  $\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} = e^{-ck}$

## Completing the Proof

We know  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn)$

$$= \lim_{n \rightarrow \infty} \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} \left(1 - \frac{i}{n}\right)^{n \ln n + cn}$$

and  $\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} = e^{-ck}$

thus  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn) = \lim_{n \rightarrow \infty} (-1)^{i+1} \frac{e^{-ci}}{i!}$

## Completing the Proof

We know  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn)$

$$= \lim_{n \rightarrow \infty} \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} \left(1 - \frac{i}{n}\right)^{n \ln n + cn}$$

and  $\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} = e^{-ck}$

thus  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn) = \lim_{n \rightarrow \infty} (-1)^{i+1} \frac{e^{-ci}}{i!} = 1 - e^{-e^{-c}}$   
 (Taylor approximation for  $1 - e^{-e^{-c}}$ ) □

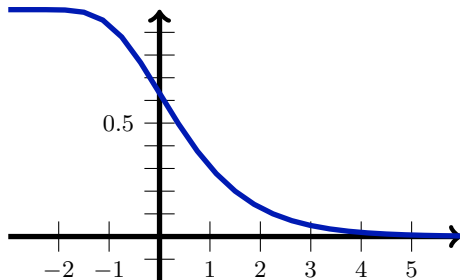
## Completing the Proof

We know  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn)$

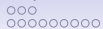
$$= \lim_{n \rightarrow \infty} \sum_{i=1}^n (-1)^{i+1} \binom{n}{i} \left(1 - \frac{i}{n}\right)^{n \ln n + cn}$$

and  $\lim_{n \rightarrow \infty} k! \cdot \binom{n}{k} \cdot \left(1 - \frac{k}{n}\right)^{n \ln n + cn} = e^{-ck}$

thus  $\lim_{n \rightarrow \infty} \text{Prob}(T > n \ln n + cn) = \lim_{n \rightarrow \infty} (-1)^{i+1} \frac{e^{-ci}}{i!} = 1 - e^{-e^{-c}}$   
(Taylor approximation for  $1 - e^{-e^{-c}}$ ) □



sharp threshold result



# Randomised Search Heuristics

# Randomised Search Heuristics

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - $(1+1)$  EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

# Randomised Search Heuristics

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - (1+1) EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

Which randomised search heuristic is the best?

# Randomised Search Heuristics

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - (1+1) EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

Which randomised search heuristic is the best?

We consider optimisation of **unknown** function  $f \in \mathcal{F}$   
with  $\mathcal{F} \subseteq \{g: S \rightarrow R\}$  **known** ( $S$  finite,  $R \subseteq \mathbb{R}$  finite)

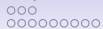
# Randomised Search Heuristics

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - (1+1) EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

Which randomised search heuristic is the best?

We consider optimisation of **unknown** function  $f \in \mathcal{F}$   
 with  $\mathcal{F} \subseteq \{g: S \rightarrow R\}$  **known** ( $S$  finite,  $R \subseteq \mathbb{R}$  finite)  
 and **black-box algorithms** for  $\mathcal{F}$ ,  
 optimising each  $f \in \mathcal{F}$  at least eventually



# A best randomised search heuristic?

## A best randomised search heuristic?

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - $(1+1)$  EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

Which randomised search heuristic is the best?

## A best randomised search heuristic?

Remember we have

- local search (first improvement, best improvement, random)
- Metropolis algorithm
- simulated annealing
- evolutionary algorithms
  - (1+1) EA
  - $(\mu+\lambda)$  EA
  - $(\mu, \lambda)$  EA
  - steady state GA
  - ...
- ...

Which randomised search heuristic is the best?

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising?

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising?    perhaps

Is it true?

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising?    perhaps

Is it true?    see proof

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

Next

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

Next

- 1 formalising deterministic black-box algorithms

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals

## Dealing with the NFL

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

### Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals
- 3 proof for deterministic search heuristics

# Dealing with the NFL

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

### Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals
- 3 proof for deterministic search heuristics
- 4 proof for randomised search heuristics

# Dealing with the NFL

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

### Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals
- 3 proof for deterministic search heuristics
- 4 proof for randomised search heuristics
- 5 putting the NFL into perspective (part 1)

# Dealing with the NFL

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

### Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals
- 3 proof for deterministic search heuristics
- 4 proof for randomised search heuristics
- 5 putting the NFL into perspective (part 1)
- 6 proof of a more general NFL

# Dealing with the NFL

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Is this surprising? perhaps

Is it true? see proof

### Next

- 1 formalising deterministic black-box algorithms
- 2 formalising optimisation goals
- 3 proof for deterministic search heuristics
- 4 proof for randomised search heuristics
- 5 putting the NFL into perspective (part 1)
- 6 proof of a more general NFL
- 7 putting the NFL into perspective (part 2)

# Describing Deterministic Black-Box Algorithms

# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$

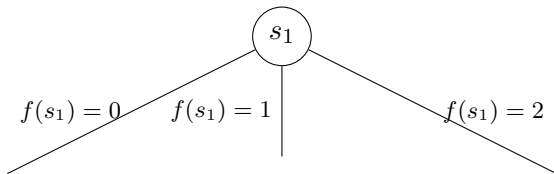
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



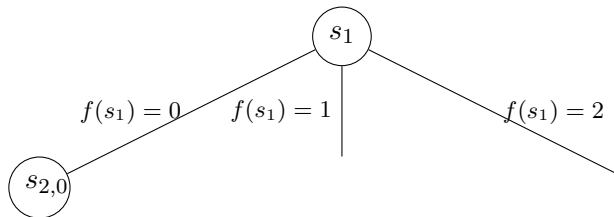
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



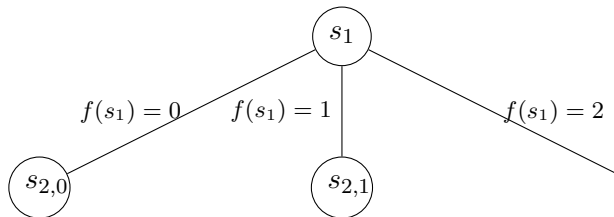
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



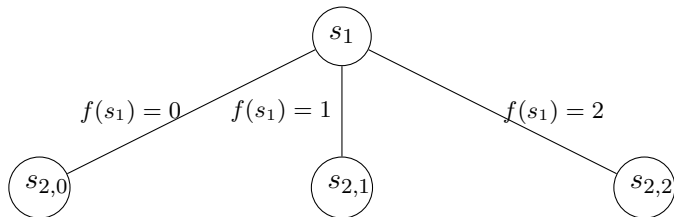
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



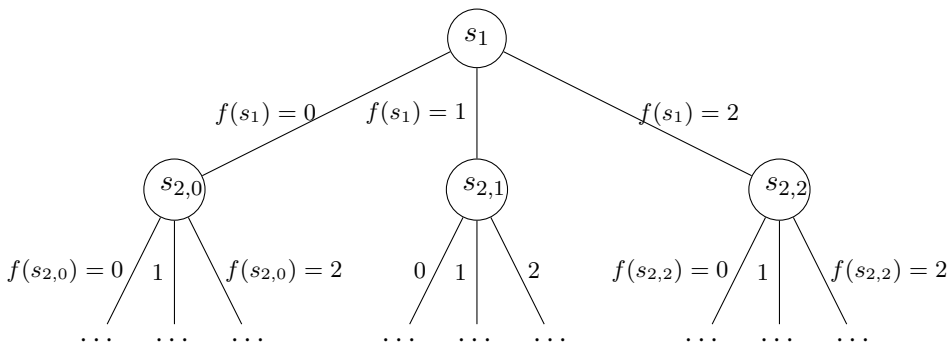
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



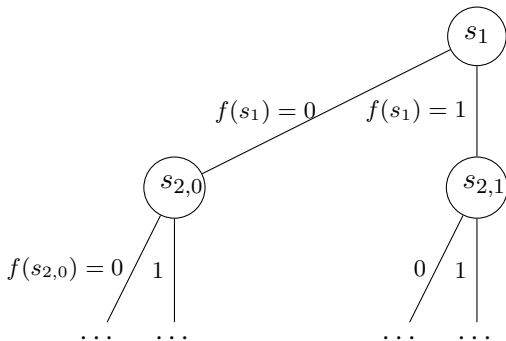
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$



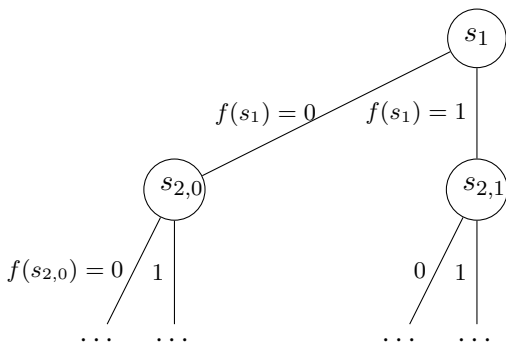
# Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$  for maximising  $f: S \rightarrow R$



## Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$  for maximising  $f: S \rightarrow R$

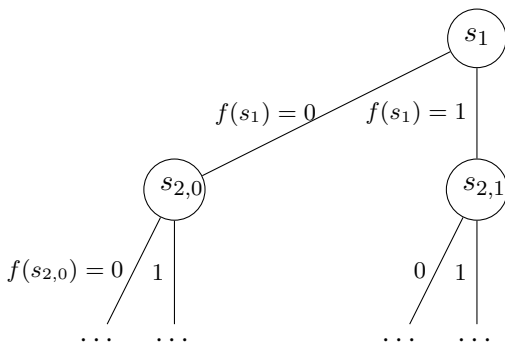


### Observations

- black-box algorithm for  $\mathcal{F} \subseteq R^S \cong |R|^S$ -ary tree with node labels from  $S$

## Describing Deterministic Black-Box Algorithms

Using the example  $R = \{0, 1, 2\}$  for maximising  $f: S \rightarrow R$



### Observations

- black-box algorithm for  $\mathcal{F} \subseteq R^S \cong |R|^S$ -ary tree with node labels from  $S$
- non-repeating black-box algorithm for  $\mathcal{F} \subseteq R^S$  has depth  $\leq |S|$

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

## Observations

- 'non-repeating' **easy to achieve** using a dictionary

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

## Observations

- 'non-repeating' **easy to achieve** using a dictionary
- NFL only counts  $f$  evaluations  $\Rightarrow$  other computational cost (including dictionary use) for free

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

## Observations

- 'non-repeating' **easy to achieve** using a dictionary
- NFL only counts  $f$  evaluations  $\Rightarrow$  other computational cost (including dictionary use) for free
- **Remember** NFL counts **distinct**  $f$  evaluations  
 $\rightsquigarrow$  'non-repeating' equivalent to not counting repetitions

Is 'finite  $S$  and  $R$ ' a strong assumption?

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

## Observations

- 'non-repeating' **easy to achieve** using a dictionary
- NFL only counts  $f$  evaluations  $\Rightarrow$  other computational cost (including dictionary use) for free
- **Remember** NFL counts **distinct**  $f$  evaluations  
 $\rightsquigarrow$  'non-repeating' equivalent to not counting repetitions

Is 'finite  $S$  and  $R$ ' a strong assumption?

## Observations

- given finite  $S$ , for any  $f$  finite  $R$  **no restriction** since  
 $|f(S)| \leq |S|$

# On Non-Repeating Black-Box Algorithms

Is 'non-repeating' a strong assumption?

## Observations

- 'non-repeating' **easy to achieve** using a dictionary
- NFL only counts  $f$  evaluations  $\Rightarrow$  other computational cost (including dictionary use) for free
- **Remember** NFL counts **distinct**  $f$  evaluations  
 $\rightsquigarrow$  'non-repeating' equivalent to not counting repetitions

Is 'finite  $S$  and  $R$ ' a strong assumption?

## Observations

- given finite  $S$ , for any  $f$  finite  $R$  **no restriction** since  
 $|f(S)| \leq |S|$
- finite  $S$  for any implementation of any  $f$  **realistic**

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree

**Note** no real change of  $A$

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree  
**Note** no real change of  $A$

**Observation**  $A$  on  $f \hat{=} \text{path in tree}$

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree  
**Note no real change** of  $A$

**Observation**  $A$  on  $f \hat{=} \text{path in tree}$   
**trace**  $T_A(f) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

# Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree  
**Note no real change** of  $A$

**Observation**  $A$  on  $f \hat{=} \text{path in tree}$   
**trace**  $T_A(f) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

**Remark** defining  $T_A(f)$  up to  $s_{|S|}$  is **simplifying notation**  
 without real change to  $A$  (as above)

## Optimisation with Deterministic Black-Box Algorithms

**Remember** deterministic black-box algorithm  $A$   
 $\hat{=} \leq |R| - 1$ -ary tree

**Simplifying Notation** deterministic black-box algorithm  $A$   
 $\hat{=} |R|$ -ary tree  
**Note no real change** of  $A$

**Observation**  $A$  on  $f \hat{=} \text{path in tree}$   
**trace**  $T_A(f) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_{|S|}, f(s_{|S|})) \rangle$

**Remark** defining  $T_A(f)$  up to  $s_{|S|}$  is **simplifying notation**  
 without real change to  $A$  (as above)

**Consider** projection of  $T_A(f)$  on function values  
 $V_A(f) = \langle f(s_1), f(s_2), \dots, f(s_{|S|}) \rangle$

# Optimisation Goals

## Remember

- deterministic black-box algorithm  $A \hat{=} |R|$ -ary tree

- $A$  on  $f \hat{=} \text{trace}$

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

- projection on function values

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

# Optimisation Goals

## Remember

- deterministic black-box algorithm  $A \hat{=} |R|$ -ary tree

- $A$  on  $f \hat{=} \text{trace}$

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

- projection on function values

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

## Useful Notation

$T_A(f, t) = \text{first } t \text{ components of } T_A(f)$

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

# Optimisation Goals

## Remember

- deterministic black-box algorithm  $A \hat{=} |R|$ -ary tree

- $A$  on  $f \hat{=} \text{trace}$

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

- projection on function values

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

## Useful Notation

$T_A(f, t) = \text{first } t \text{ components of } T_A(f)$

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

$V_A(f, t) = \text{first } t \text{ components of } V_A(f)$

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

# Optimisation Goals

## Remember

- deterministic black-box algorithm  $A \hat{=} |R|$ -ary tree

- $A$  on  $f \hat{=} \text{trace}$

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

- projection on function values

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

**Useful Notation**

$T_A(f, t) = \text{first } t \text{ components of } T_A(f)$   
 $T_A(f, t) = \langle (s_1, f(s_1)), (s_2, f(s_2)), \dots, (s_t, f(s_t)) \rangle$   
 $V_A(f, t) = \text{first } t \text{ components of } V_A(f)$   
 $V_A(f, t) = \langle f(s_1), f(s_2), \dots, f(s_t) \rangle$

## Definition (Optimisation Goal)

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

# Proving the NFL — Part 1

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

# Proving the NFL — Part 1

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Now **proof** for **deterministic** black-box algorithms

## Proving the NFL — Part 1

### NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Now **proof** for **deterministic** black-box algorithms

**Need to prove** For all deterministic black-box algorithms  $A, A'$  and all optimisation goals  $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

# Proving the NFL — Part 1

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Now **proof** for **deterministic** black-box algorithms

**Need to prove** For all deterministic black-box algorithms  $A, A'$  and all optimisation goals  $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

$$\frac{\sum_{f \in R^S} M(V_A(f, |S|))}{|R^S|} = \frac{\sum_{f \in R^S} M(V_{A'}(f, |S|))}{|R^S|}$$

# Proving the NFL — Part 1

## NFL Theorem (1996)

For any finite  $S$  and  $R$ , on average, all black-box algorithms for  $\mathcal{F} = R^S$  make an equal number of distinct  $f$  evaluations until an arbitrary optimisation goal is reached.

Now **proof** for **deterministic** black-box algorithms

**Need to prove** For all deterministic black-box algorithms  $A, A'$  and all optimisation goals  $M: \{V_A(f, t) \mid A, f, t\} \rightarrow \mathbb{R}$

$$\frac{\sum_{f \in R^S} M(V_A(f, |S|))}{|R^S|} = \frac{\sum_{f \in R^S} M(V_{A'}(f, |S|))}{|R^S|}$$

**Sub-Claim 1**  $\forall \text{det. BBA } A, f, g \in R^S:$   
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall$  det. BBA  $A, f, g \in R^S$  :  
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall$  det. BBA  $A, f, g \in R^S$  :  
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

### Proof

**Let**  $A, f, g$  be fixed arbitrarily with  $V_A(f, |S|) = V_A(g, |S|)$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall$  det. BBA  $A, f, g \in R^S$  :  
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

### Proof

**Let**  $A, f, g$  be fixed arbitrarily with  $V_A(f, |S|) = V_A(g, |S|)$

**Observation**  $T_A(f, 1) = T_A(g, 1)$  since  $V_A(f, 1) = V_A(g, 1) = v_1$   
 and  $s_1$  depends only on  $A$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall \text{det. BBA } A, f, g \in R^S :$   
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

### Proof

**Let**  $A, f, g$  be fixed arbitrarily with  $V_A(f, |S|) = V_A(g, |S|)$

**Observation**  $T_A(f, 1) = T_A(g, 1)$  **since**  $V_A(f, 1) = V_A(g, 1) = v_1$   
 and  $s_1$  depends only on  $A$

**Observation**  $T_A(f, 2) = T_A(g, 2)$  **since**  $V_A(f, 2) = V_A(g, 2)$   
 and  $s_2$  depends only on  $A, s_1$  and  $v_1$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall \text{det. BBA } A, f, g \in R^S :$   
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

### Proof

**Let**  $A, f, g$  be fixed arbitrarily with  $V_A(f, |S|) = V_A(g, |S|)$

**Observation**  $T_A(f, 1) = T_A(g, 1)$  **since**  $V_A(f, 1) = V_A(g, 1) = v_1$   
 and  $s_1$  depends only on  $A$

**Observation**  $T_A(f, 2) = T_A(g, 2)$  **since**  $V_A(f, 2) = V_A(g, 2)$   
 and  $s_2$  depends only on  $A, s_1$  and  $v_1$

**Observation** **in the same way**  $T_A(f, t) = T_A(g, t)$   
 for all  $t \in \{1, 2, \dots, |S|\}$

## Proving Sub-Claim 1

**Sub-Claim 1**  $\forall \text{det. BBA } A, f, g \in R^S :$   
 $V_A(f, |S|) = V_A(g, |S|) \Rightarrow f = g$

### Proof

**Let**  $A, f, g$  be fixed arbitrarily with  $V_A(f, |S|) = V_A(g, |S|)$

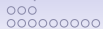
**Observation**  $T_A(f, 1) = T_A(g, 1)$  **since**  $V_A(f, 1) = V_A(g, 1) = v_1$   
 and  $s_1$  depends only on  $A$

**Observation**  $T_A(f, 2) = T_A(g, 2)$  **since**  $V_A(f, 2) = V_A(g, 2)$   
 and  $s_2$  depends only on  $A, s_1$  and  $v_1$

**Observation** **in the same way**  $T_A(f, t) = T_A(g, t)$   
 for all  $t \in \{1, 2, \dots, |S|\}$

**Thus**  $f = g$  ✓





# Summary & Take Home Message

Things to remember



# Summary & Take Home Message

## Things to remember

- coupon collector

# Summary & Take Home Message

## Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

# Summary & Take Home Message

## Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

## Take Home Message

# Summary & Take Home Message

## Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

## Take Home Message

- Even simple problems can require some amount of analysis.

# Summary & Take Home Message

## Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

## Take Home Message

- Even simple problems can require some amount of analysis.
- Probability theory yields precise and useful answers to practically relevant problems.

## Summary & Take Home Message

### Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

### Take Home Message

- Even simple problems can require some amount of analysis.
- Probability theory yields precise and useful answers to practically relevant problems.
- When making too general statements about randomised search heuristics the statements are either trivial or false.

## Summary & Take Home Message

### Things to remember

- coupon collector
- NFL: All search heuristics perform equal over the class of all functions.

### Take Home Message

- Even simple problems can require some amount of analysis.
- Probability theory yields precise and useful answers to practically relevant problems.
- When making too general statements about randomised search heuristics the statements are either trivial or false.
- One needs to consider the class of optimisation problems to achieve good performance.