No Free Lunch

PART I

Michael D. Vose

The University of Tennessee

PART II

Darrell Whitley

Colorado State University

GECCO'07, July 7-11, 2007, London, England, United Kingdom.
ACM 978-1-59593-698-1/07/0007.

Vose Notes: from "NO FREE LUNCH..." (by Chris Schumacher)

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function between finite sets, and let y_i denote $f(x_i)$. The domain $\mathcal{X}$ and co-domain $\mathcal{Y}$ are fixed for this discussion, but the function f may vary.

Definition A *trace* T is a sequence of pairs $\langle (x_0, y_0), \ldots, (x_{m-1}, y_{m-1}) \rangle$ where the x components are unique. The following notation will be used

$$\begin{array}{rcll} \epsilon & = & \langle\,\rangle & \text{(the empty trace)} \\ T_x & = & \langle x_0, x_1, \ldots, x_{m-1} \rangle & \text{(sequence of } x \text{ components)} \\ T_y & = & \langle y_0, y_1, \ldots, y_{m-1} \rangle & \text{(sequence of } y \text{ components)} \\ \{T\} & = & \{(x_0, y_0), \ldots, (x_{m-1}, y_{m-1})\} & \text{(the set of components of } T\text{)} \end{array}$$

Definition A trace T is *complete* if $T_x = \mathcal{X}$, and in that case $\{T\} = f$. The set of all non-complete traces corresponding to a function f is denoted by $\mathcal{T}(f)$. Define $\mathcal{T}$ by

$$\mathcal{T} = \bigcup_f \mathcal{T}(f)$$

Definition A *search operator* is a function $g : \mathcal{T} \to \mathcal{X}$ which maps a non-complete trace T to some element not contained in T_x.

Definition A *deterministic non-repeating Black Box search algorithm* A corresponds to a search operator g, and will be referred to simply as a *search algorithm*. Algorithm A applied to function f is denoted by A_f, it takes argument $T \in \mathcal{T}(f)$ and returns

$$T \parallel (g(T), f \circ g(T))$$

where $\parallel$ is the concatenation operator.

Thus A_f is a function which extends non-complete traces. Search algorithms are thought of as operating in discrete steps, beginning from the empty trace. The first two steps of A_f are

$$\begin{array}{rcl} T & = & A_f(\epsilon) = \epsilon \parallel (g(\epsilon), f \circ g(\epsilon)) \\ T' & = & A_f(T) = T \parallel (g(T), f \circ g(T)) \end{array}$$

Multiple steps are abbreviated in the usual way,

$$A_f^m = \underbrace{A_f \circ \cdots \circ A_f}_{m \text{ times}}$$

(and A_f^0 is the identity function). To streamline notation, the complete trace $A_f^{|\mathcal{X}|}(\epsilon)$ generated by search algorithm A applied to function f will be denoted by $A(f)$. In particular, $\{A(f)\} = f$.

Search algorithms A and B are considered identical if and only if they generate the same complete trace for all f,

$$A(f) = B(f)$$

Definition A *search path* is a sequence of unique elements from $\mathcal{X}$. The *search path associated with trace* T is T_x.

Definition A *performance vector* is a sequence of values from $\mathcal{Y}$. The *performance vector associated with trace* T is T_y.

Search algorithm A applied to function f generates the search path $A(f)_x$ and the performance vector $A(f)_y$. The search operator g — to which A corresponds — determines how the space is explored; it generates the x components of the trace $A(f)$. The function f determines utility; it produces the y components of the trace $A(f)$.

Definition A *trace table* is a table whose rows are labeled by the algorithms and whose columns are labeled by the functions; the element in row A and column f is $A(f)$.

A trace table contains complete descriptions of all search algorithms applied to all functions. The trace table for $\mathcal{X} = \{0, 1, 2\}$, $\mathcal{Y} = \{0, 1\}$, where trace $\langle (a, x), (b, y), (c, z) \rangle$ is represented as $\begin{smallmatrix} abc \\ xyz \end{smallmatrix}$, is

	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
A_0	012	012	012	012	012	012	012	012
	000	100	010	110	001	101	011	111
A_1	021	021	021	021	021	021	021	021
	000	100	001	101	010	110	011	111
A_2	012	021	012	021	012	021	012	021
	000	100	010	101	001	110	011	111
A_3	021	012	021	012	021	012	021	012
	000	100	001	110	010	101	011	111
A_4	102	102	102	102	102	102	102	102
	000	010	100	110	001	011	101	111
A_5	120	120	120	120	120	120	120	120
	000	001	100	101	010	011	110	111
A_6	120	120	102	102	120	120	102	102
	000	001	100	110	010	011	101	111
A_7	102	102	120	120	102	102	120	120
	000	010	100	101	001	011	110	111
A_8	201	201	201	201	201	201	201	201
	000	010	001	011	100	110	101	111
A_9	210	210	210	210	210	210	210	210
	000	001	010	011	100	101	110	111
A_{10}	210	210	210	210	201	201	201	201
	000	001	010	011	100	110	101	111
A_{11}	201	201	201	201	210	210	210	210
	000	010	001	011	100	101	110	111

Definition A *performance table* is a table whose rows are labeled by the algorithms and whose columns are labeled by the functions; the element in row A and column f is $A(f)_y$.

Theorem 1 (Uniqueness) *No row of a performance table contains any element more than once.*

Proof by contradiction: assume $A(f)_y = A(f')_y$ and induct on i to prove $A_f^i(\epsilon) = A_{f'}^i(\epsilon)$. Thus $A(f) = A(f')$ and $f = f'$.

The base case ($i = 0$) is trivial: $A_f^0(\epsilon) = \epsilon = A_{f'}^0(\epsilon)$.
For the inductive case, note that

$$\begin{aligned} A_f^{i+1}(\epsilon) &= A_f(A_f^i(\epsilon)) &= A_f^i(\epsilon) \parallel (x, f(x)) \\ A_{f'}^{i+1}(\epsilon) &= A_{f'}(A_{f'}^i(\epsilon)) &= A_{f'}^i(\epsilon) \parallel (x', f'(x')) \end{aligned}$$

where $x = g(A_f^i(\epsilon))$ and $x' = g(A_{f'}^i(\epsilon))$. It follows that $x = x'$, since $A_f^i(\epsilon) = A_{f'}^i(\epsilon)$ by the inductive hypothesis. Moreover, $f(x) = f'(x')$ since $A(f)_y = A(f')_y$ by assumption. Hence $A_f^{i+1}(\epsilon)$ and $A_{f'}^{i+1}(\epsilon)$ are constructed by extending the same trace with the same ordered pair (which completes the inductive proof). □

Theorem 2 (Invariance) *Any two rows in a performance table are permutations of each other.*

Proof: The number of possible performance vectors is $|\mathcal{Y}|^{|\mathcal{X}|}$ (there are $|\mathcal{Y}|$ ways to choose each of the $|\mathcal{X}|$ elements of a performance vector), and this is also the number of entries in a row of the table (there are $|\mathcal{Y}|^{|\mathcal{X}|}$ functions). By the uniqueness theorem, no performance vector is repeated in any row, and thus each possible performance vector occurs exactly once in each row. □

Definition A *performance measure with respect to a set F of functions* is any function μ_F which is defined over the collection of all algorithms such that $\mu_F(A)$ is a function of $\{A(f)_y : f \in F\}$. Two algorithms *perform equally well on F* if they are evaluated identically by every performance measure with respect to F.

Definition An *overall performance measure* is a performance measure with respect to the set of all functions. Two algorithms *perform on average equally well* if they are evaluated identically by every overall performance measure.

Theorem 3 (NFL) *Each deterministic non-repeating Black Box search algorithm performs on average equally well.*

Proof: From the invariance theorem, each row in a performance table represents the same set of performance vectors, and thus any two algorithms are evaluated based on the exact same data for computing performance. □

Sharpening NFL

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function and let $\sigma : \mathcal{X} \to \mathcal{X}$ be a permutation (i.e., σ is one-to-one and onto).

Definition The *permutation* σf *of* f is the function $\sigma f : \mathcal{X} \to \mathcal{Y}$ defined by $\sigma f(x) = f(\sigma^{-1}(x))$.

Theorem 4 *If* T *and* T' *are complete traces, then* $T_y = T'_y \implies \{T'\}$ *is a permutation of* $\{T\}$.

Proof: Let σ map the ith component of T_x (call it k) to the ith component of T'_x (call it j) . Then $\sigma^{-1}(j) = k$, and

$$\sigma\{T\}(j) \;=\; \{T\}(k) \;=\; (T_y)_i \;=\; (T'_y)_i \;=\; \{T'\}(j)$$

As i varies, j ranges over all of $\mathcal{X}$. □

Definition Let A be an algorithm corresponding to search operator g. The *permutation* σA *of* A is the algorithm with search operator σg defined by $\sigma g(T) = \sigma^{-1}(g(\sigma_x(T)))$ where σ_x operates on the x values of trace T by applying σ to each of them while leaving the y values untouched, and where $\sigma_x(\epsilon) = \epsilon$.

Theorem 5 (Duality) $(\sigma A)(f)_y = A(\sigma f)_y$

Proof: Induct on i to show $\sigma_x((\sigma A)^i_f(\epsilon)) = A^i_{\sigma f}(\epsilon)$. Thus $\sigma_x((\sigma A)(f)) = A(\sigma f)$ which proves the theorem (σ_x does not change the y values in a trace).

The base case ($i = 0$) is $\sigma_x(\epsilon) = \epsilon$.
For the inductive case, note that

$$\begin{aligned}
\sigma g((\sigma A)^i_f(\epsilon)) &= \sigma^{-1}\!\circ g(\sigma_x((\sigma A)^i_f(\epsilon))) \\
&= \sigma^{-1}\!\circ g(A^i_{\sigma f}(\epsilon)) \\
f\circ\sigma g((\sigma A)^i_f(\epsilon)) &= f(\sigma^{-1}\!\circ g(A^i_{\sigma f}(\epsilon))) \\
&= \sigma f(g(A^i_{\sigma f}(\epsilon)))
\end{aligned}$$

by the inductive hypothesis. Therefore

$$\begin{aligned}
(\sigma A)^{i+1}_f(\epsilon) &= (\sigma A)^i_f(\epsilon) \;||\; (\sigma g((\sigma A)^i_f(\epsilon)), f\circ\sigma g((\sigma A)^i_f(\epsilon))) \\
&= (\sigma A)^i_f(\epsilon) \;||\; (\sigma^{-1}\!\circ g(A^i_{\sigma f}(\epsilon)), \sigma f(g(A^i_{\sigma f}(\epsilon)))) \\
\sigma_x((\sigma A)^{i+1}_f(\epsilon)) &= \sigma_x((\sigma A)^i_f(\epsilon)) \;||\; (g(A^i_{\sigma f}(\epsilon)), \sigma f(g(A^i_{\sigma f}(\epsilon)))) \\
&= A^i_{\sigma f}(\epsilon) \;||\; (g(A^i_{\sigma f}(\epsilon)), \sigma f\circ g(A^i_{\sigma f}(\epsilon))) \\
&= A^{i+1}_{\sigma f}(\epsilon)
\end{aligned}$$

(using the inductive hypothesis again). □

Definition A set F of functions is *closed under permutation* if for every permutation σ,

$$f \in F \implies \sigma f \in F$$

Definition *No free lunch holds over* F if every algorithm performs equally well on F.

Theorem 6 (Sharpened NFL) No free lunch holds over a set F of functions if and only if F is closed under permutation.

Proof: Suppose F is closed under permutation. If for arbitrary algorithms A and A' the sets $S = \{A(f)_y : f \in F\}$ and $S' = \{A'(h)_y : h \in F\}$ are equal, then any two algorithms are evaluated based on the same data for computing performance, and no free lunch holds over F.

By the invariance theorem, given any f there exists h such that $A(f)_y = A'(h)_y$. It follows that h is a permutation of f (Theorem 4). Therefore $f \in F \Longrightarrow h \in F$. Hence $S \subset S'$. The reverse containment follows by symmetry.

Conversely, assume by way of contradiction that no free lunch holds over F, but that F is not closed under permutation; let σ be such that $f \in F$ and $\sigma f \notin F$.

Fix an algorithm $\mathcal{A}$ and consider the performance measure

$$\mu_F(A) \ = \ [\mathcal{A}(f)_y \in \{A(h)_y \ : \ h \in F\}]$$

where [*expression*] is 1 if *expression* is true, and 0 otherwise. Since $\mu_F(\mathcal{A}) = 1$ and no free lunch holds over F, it must happen that $\mu_F(A) = 1$ for the particular choice $A = \sigma^{-1}\mathcal{A}$. Therefore,

$$\mathcal{A}(f)_y \in \{(\sigma^{-1}\mathcal{A})(h)_y \ : \ h \in F\}$$

which leads to a contradiction as follows. Appealing to the duality theorem,

$$\{(\sigma^{-1}\mathcal{A})(h)_y \ : \ h \in F\} = \{\mathcal{A}(\sigma^{-1}h)_y \ : \ h \in F\} = \{\mathcal{A}(h)_y \ : \ \sigma h \in F\}$$

Appealing to the uniqueness theorem, $\mathcal{A}(f)_y \in \{\mathcal{A}(h)_y \ : \ \sigma h \in F\} \Longrightarrow \ \sigma f \in F$ □

Discussion

Some "real world algorithms" revisit points in the search space, and so a description of how they explore $\mathcal{X}$ would not correspond to a trace. If $\mathcal{R}$ were such an algorithm, one could consider the related non-repeating algorithm A which behaves exactly as does $\mathcal{R}$ with the exception that it refrains from visiting any point of $\mathcal{X}$ more than once. If some version of NFL applies to A, then one might use the close relationship between A and $\mathcal{R}$ to make inferences concerning $\mathcal{R}$.

Some "real world algorithms" are stochastic, rather than deterministic. This is often a fiction, however, since most random number generators are in fact deterministic. Hence a "stochastic algorithm" $\mathcal{R}$ based on a deterministic random number generator with seed s is in reality a family $\mathcal{R}(s)$ of deterministic algorithms parametrized by s. Some version of NFL may apply to every member of that parametrized family of deterministic algorithms.

Given a deterministic non-repeating algorithm $\mathcal{A}$, one might wonder whether it was in fact a *deterministic non-repeating Black Box search algorithm*. The issue is whether it corresponds to a search operator g as described on page one. The answer is yes, provided the manner in which $\mathcal{A}$ explores $\mathcal{X}$ at step t is a function of the trace T describing its past exploration at previous steps (defining g to map T to whatever point $\mathcal{A}$ explores at step t makes $\mathcal{A}$ correspond to g).

No Free Lunch: 1995-2006

Darrell Whitley
Colorado State University

NFL: No Free Lunch

All search algorithms are equivalent when compared over all possible discrete functions.

Wolpert, Macready (1995)
No free lunch theorems for search. Santa Fe Institute.

Radcliffe, Surry (1995)
Fundamental Limitations on Search Algorithms: Springer Verlag LNCS 1000.

No Free Lunch for Gray and Binary

All search algorithms are equivalent when compared over all possible representations.

Variations on No Free Lunch

For ANY measure of algorithm performance:

The aggregate behavior of any two search algorithms is equivalent when compared all possible discrete functions.

The aggregate behavior of ALL possible search algorithms is equivalent when compared over any two discrete functions.

At each distinct "iteration" of search
the aggregate behavior of all possible search algorithms is IDENTICAL at each and every iteration.

Variations on No Free Lunch

Consider any algorithm A_i applied to function f_j.

On(A_i, f_j) outputs the order in which A_i visits the elements in the codomain of f_j. For every pair of algorithms A_k and A_i and for any function f_j, there exist a function f_l such that

$$On(A_i, f_j) \equiv On(A_k, f_l)$$

Consider a "BestFirst" local search with restarts.

Consider a "WorstFirst" local search with restarts.

For every j there exists an l such that

$$On(BestFirst, f_j) \equiv On(WorstFirst, f_l)$$

ENUMERATION is a search algorithm.

Thus, No Free Lunch implies that on average,
no search algorithm is better than enumeration.

Furthermore, because bias in search algorithms causes them to focus the search, most are prone to resampling.

If resampling is considered,
“focused” search algorithms are WORSE than enumeration

NFL IGNORES RESAMPLING

An algorithm is modeled as a permutation
representing the order in which new points are tested.

Behavior is defined in terms of the evaluation function output
which defines the co-domain of the function.

Assume that one is given a fixed set of co-domain values.
Set of Functions = Set of Permutations.

BEHAVIORS	FUNCTIONS
A1: 1 2 3	F1: A B C
A2: 1 3 2	F2: A C B
A3: 2 1 3	F3: B A C
A4: 2 3 1	F4: B C A
A5: 3 1 2	F5: C A B
A6: 3 2 1	F6: C B A

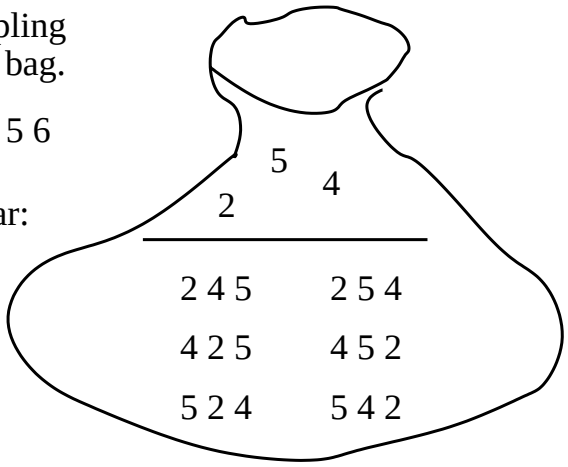
Assume $(A > B)\&(B > C)$.
Take 2 steps, return the maximum found.

	F1	F2	F3	F4	F5	F6
A1	A	A	A	B	A	B
A2	A	A	B	A	B	A
A3	A	A	A	B	A	B
A4	B	B	A	A	A	A
A5	A	A	B	A	B	A
A6	B	B	A	A	A	A

NFL is just like sampling
from a grab bag.

Co−Domain: 1 2 3 4 5 6

Values sampled so far:
3 6 1 ....



Theorem:
NFL holds for a set of functions IFF
the set of functions form a permutation set.

The “Permutation Set” is the closure of a set
of functions with respect to a permutation operator.
(Schmacher, Vose and Whitley–GECCO 2001;
also, see Rawlins 1991, Radcliffe and Surry 1995)

```
F1:  0 0 1 2      F7:  0 2 0 1
F2:  0 1 0 2      F8:  0 2 1 0
F3:  1 0 0 2      F9:  1 2 0 0

F4:  0 0 2 1     F10:  2 0 0 1
F5:  0 1 2 0     F11:  2 0 1 0
F6:  1 0 2 0     F12:  2 1 0 0
```

OBSERVATION: The Union of Permutation Sets is also a Permutation Set. The sampling probability can be different across Permutation Sets.

Sampling Need not be Uniform

```
F1:  A B C   12/100        F1:  0 0 0 1   7/100
F2:  A C B   12/100        F2:  0 0 1 0   7/100
F3:  B A C   12/100        F3:  0 1 0 0   7/100
F4:  B C A   12/100        F4:  1 0 0 0   7/100
F5:  C A B   12/100
F6:  C B A   12/100
```


Machine Learning and NFL

1	1	1	1
1			1
0			0
0	0	0	0

```
L1   ALL   HD          L2   ALL   HD
==   ========          ==   ========
     00    0                00    1
00   01    1           10   01    2
     10    1                10    0
     11    2                11    1
```

Theorem:
Given a finite set of N unique co-domain values, NFL hold over a set of N! functions where the average description length is O(N log N).

Sketch of Proof:
Construction a Binary Tree with N! leaves. Each leaf represents one of the N! functions. To just label each function requires log(N!) bits. Each label has average length log(N!) = O(N log N).
Note enumeration also has cost O(N log N).

Corollary:
If a fixed fraction of the co-domain values are unique, the set of N! functions where NFL holds has average description length O(N log N).

NFL holds over sets with 1 member.

```
F = 0 0 0 0
```

NFL holds over needle-in-a-haystack functions.

```
F1 = 0 0 0 1
F2 = 0 0 1 0
F3 = 0 1 0 0
F4 = 1 0 0 0
```

The set of Binary strings is a permutation set

```
0 0 0 0                                   1 1 1 1

0 0 0 1          0 0 1 1          1 1 1 0
0 0 1 0          0 1 0 1          1 1 0 1
0 1 0 0          1 0 0 1          1 0 1 1
1 0 0 0          0 1 1 0          0 1 1 1
                 1 0 1 0
                 1 1 0 0
```


Let $P(F)$ compute the permutation closure of F, where F is a set of functions.

Let $K = |P(F)|$.

Then the average description length needed to distinguish the members of that set is $lg(K)$.

If $lg(K)$ is exponential, then the permutation set is $uncompressible$.

If $lg(K)$ is polynomial, then the permutation set is $compressible$.

QUESTION:
How should we evaluate search algorithms?

Let β represent a set of benchmarks. $P(\beta)$ is the permutation closure over β.

If algorithm **S** *is better than algorithm* **T** *on* β
THEN **T** *is better than* **S** *on* $P(\beta) - \beta$.


```
                          Algorithm 1    Algorithm 2
          F1:  1 2 3    f(1) --> 1     f(3) --> 3
Set A     F2:  1 3 2    f(1) --> 1     f(3) --> 2

          F3:  2 1 3    f(1) --> 2     f(3) --> 3
          F4:  2 3 1    f(1) --> 2     f(3) --> 1
          F5:  3 1 2    f(1) --> 3     f(3) --> 2
Set B     F6:  3 2 1    f(1) --> 3     f(3) --> 1
```

	Algorithm 1	Algorithm 2	Difference
Set A	2	5	3
Set B	10	7	3

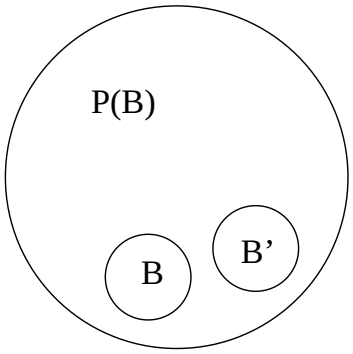
The cumulative difference must be the same

```
                          Algorithm 1    Algorithm 2
          F1:  1 2 3    f(1) --> 1     f(3) --> 3
Set A     F2:  1 3 2    f(1) --> 1     f(3) --> 2

          F3:  2 1 3    f(1) --> 2     f(3) --> 3
          F4:  2 3 1    f(1) --> 2     f(3) --> 1
          F5:  3 1 2    f(1) --> 3     f(3) --> 2
Set B     F6:  3 2 1    f(1) --> 3     f(3) --> 1
```

	Algorithm 1	Algorithm 2	Difference
Set A	1	2.5	1.5
Set B	2.5	1.75	0.75

Average difference is not the same



Given algorithms **S** and **T** we know

$$On(S, f_j \in \beta) \equiv On(T, f_l)$$

thus we can construct another test set β' such that

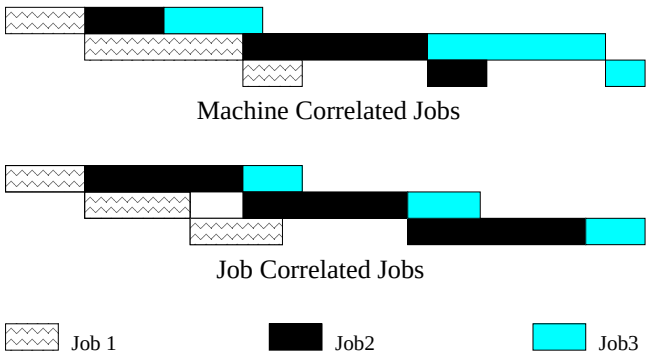
$$f_j \in \beta \rightarrow f_l \in \beta'$$

The behavior of **T** *on* β' *is IDENTICAL to the behavior of* **S** *on* β*.*

NO FREE LUNCH does not hold over the class of problems in NP; they are not black box optimization problems.

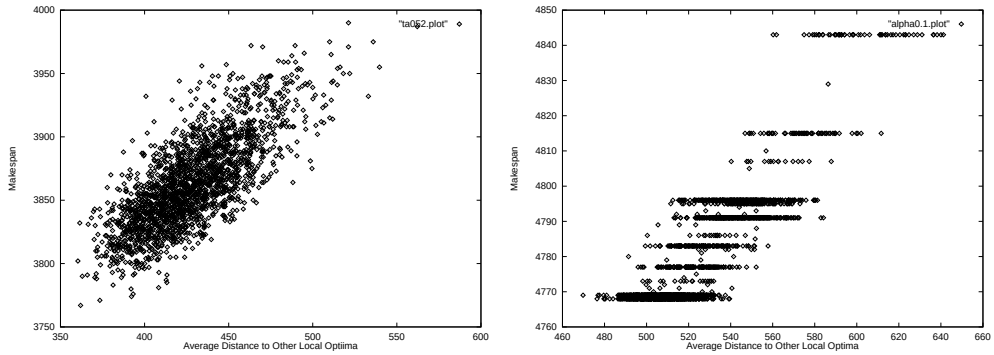
For example, some problems in NP that have ratio bounds which can be exploited by branch and bound algorithms.

Even, so claims about which algorithms apply to which problems is a concern.

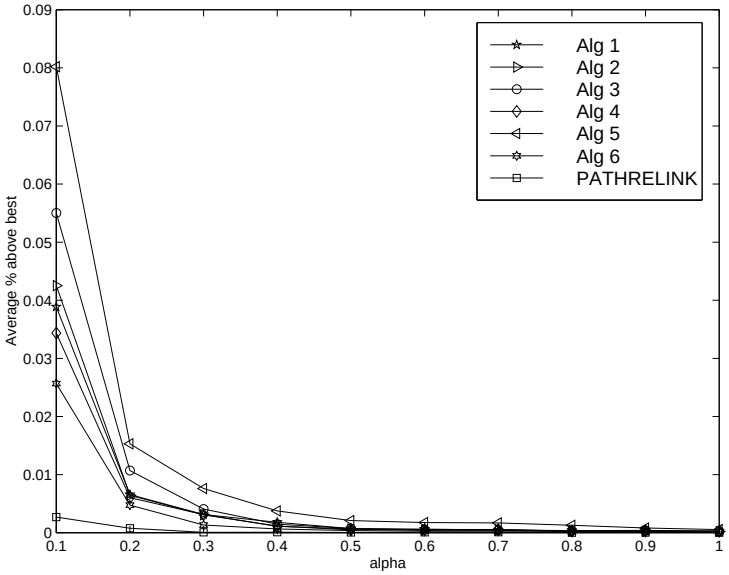


The PERMUTATION FLOWSHOP SCHEDULING PROBLEM.

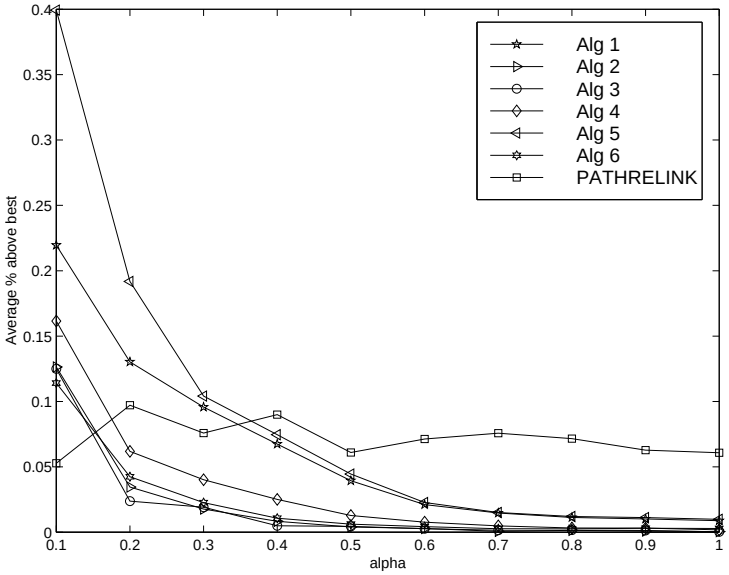
Benchmark are typically generated randomly. Real-world problems may have correlated structure. Job could be *machine correlated* or *job correlated*.



GECCO-2006 –23



JOB CORRELATED PROBLEMS. Performance of optimization algorithms. The degree of randomness is indicated along the x-axis, while the deviation from the best-known solution is indicated along the y-axis.



MACHINE CORRELATED PROBLEMS. Performance of optimization algorithms. The degree of randomness is indicated along the x-axis, while the deviation from the best-known solution is indicated along the y-axis.

S. Christensen and F. Oppacher
What can we learn from No Free Lunch? GECCO 2001

A SUBMEDIAN-SEEKER Type Algorithm

1. Evaluate a sample of points and estimate median(f).
2. If $f(x_i) < median(f)$ then sample a neighbor of x_i.
   Else sample a new random point.
3. Repeat step 2 until half of space is explored.

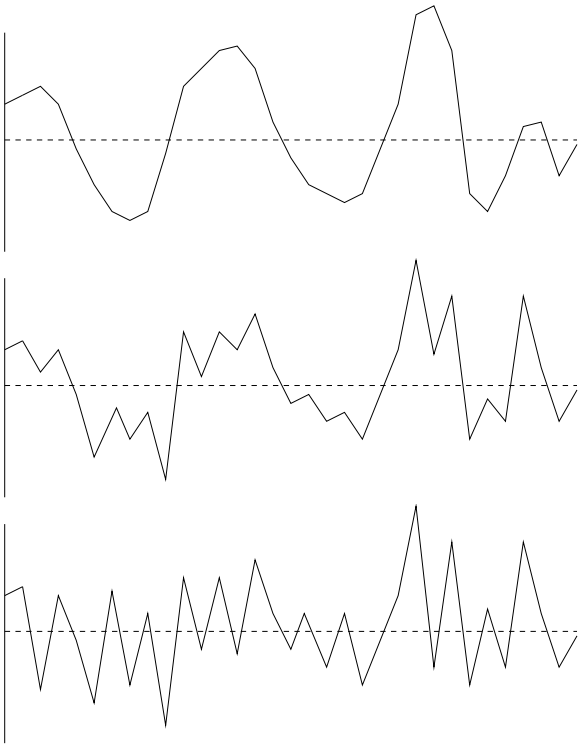
Assume f is 1-dimensional, a bijection, and we know $median(f)$.

Let $M(f)$ measures the number of submedian values of f
that have *supermedian successors*.

There exists M_{crit} such that when $M(f) < M_{crit}$
SubMedian-Seeker is better than random search.

SUBMEDIAN-SEEKER beats random enumeration when:

1. f is a uniformly sample polynomial of degree at most k and $M_{crit} > k/2$
2. f is a truncated Fourier series of at most k harmonics uniformly sampled over [0,1) at n locations and $M_{crit} > k/2$
3. Each extremum of f is represented by at least 6 points on average



Structure is Important

Random Number Generators produce functions that are in some restricted sense compressible. But they are designed to have minimal structure.

Consider "WorstFirst" local search again.
For every j there exists an l such that

$$On(BestFirst, f_j) \equiv On(WorstFirst, f_l)$$

There are "structured functions" that do not fit our usual notion of being "searchable."

NO FREE LUNCH and REPRESENTATION

Radcliffe, Surry (1995) Fundamental Limitations on Search Algorithms: Springer Verlag LNCS 1000.

The behavior of any two algorithms are identical over all possible representations of a single function.

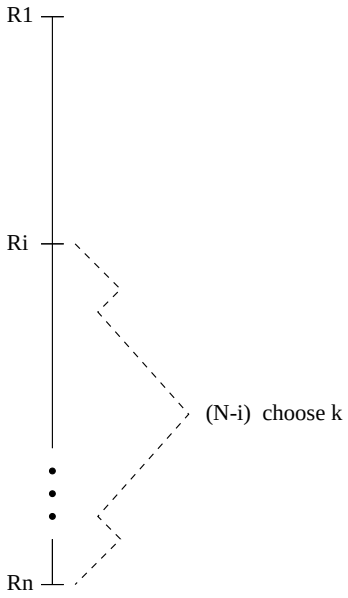
"NO-FREE-LUNCH-like" results

The behavior of any two algorithms are identical over over the set of Gray and the set of Binary representations over all possible functions.

Counting Local Optima

The probability that string i is a local minimun under an arbitrary transformation of a k-neighborhood search space is:

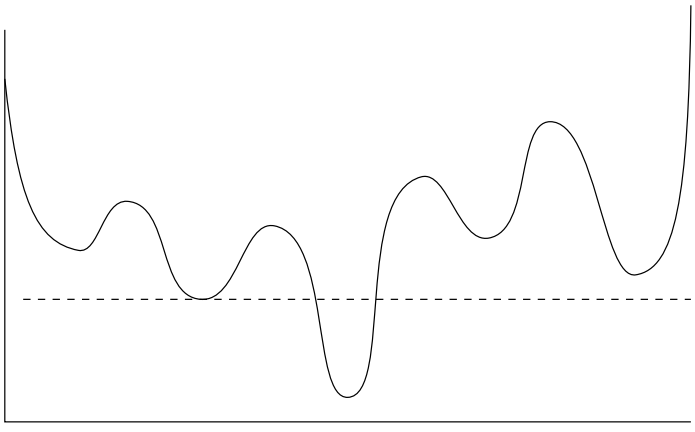
$$P(i) = \frac{\binom{N-i}{k}}{\binom{N-1}{k}} \quad [1 \leq i \leq (N-k)] \tag{1}$$

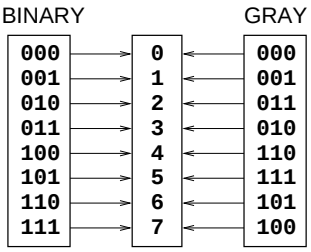


The average number of local optima over all possible representations using a k-neighbor search:

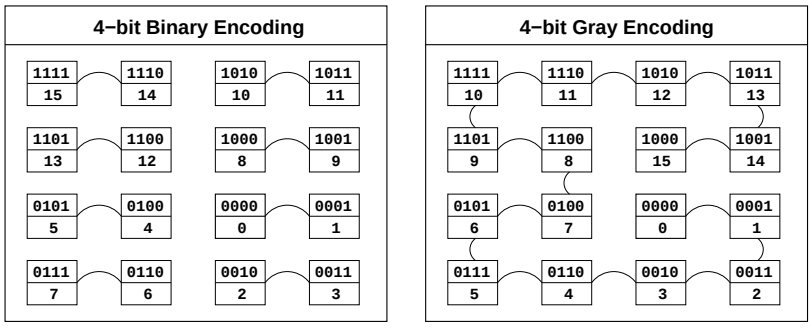
$$\mu(N,k) = \sum_{i=1}^{N-k} P(i) \tag{2}$$

$$\mu(N,k) = N/(k+1) \tag{3}$$

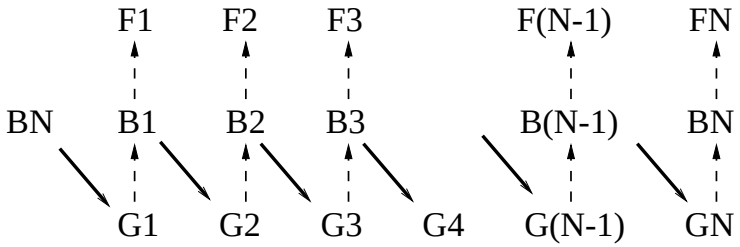




	Gray Matrix	Degray Matrix
3-bits	$\begin{vmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$
5-bits	$\begin{vmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$



”NO-FREE-LUNCH-like” results hold over
very small sets of functions for Gray and Binary representations.



The length of this “chain” is at most 2L.


```
R1: 000   001   010   011   100   101   110   111
R2: 000   001   011   010   110   111   101   100
R3: 000   001   010   011   101   100   111   110
R4: 000   001   011   010   111   110   100   101
R5: 000   001   010   011   100   101   110   111
```

Consider the integer-adjacency neighborhood.

1, 2, 3, 4, 5, 6, 7, 8, ... N-3, N-2, N-1, N

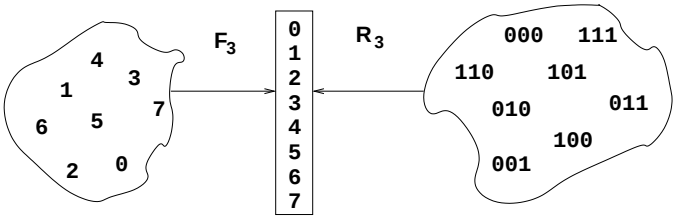
We consider a WRAPPING Neighborhood
where 1 and N are neighbors.

(We can also consider a NON-WRAPPED Neighborhood,
where 1 and N are not neighbors).

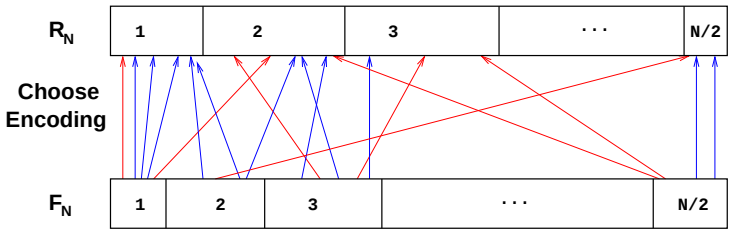
FOR WRAPPING FUNCTIONS			
K	#F K Min	# of Min Gray	# of Min Binary
1	512	512	1,024
2	14,592	23,040	27,776
3	23,040	49,152	48,896
4	2,176	7,936	2,944
Sum	40,320	80,640	80,640

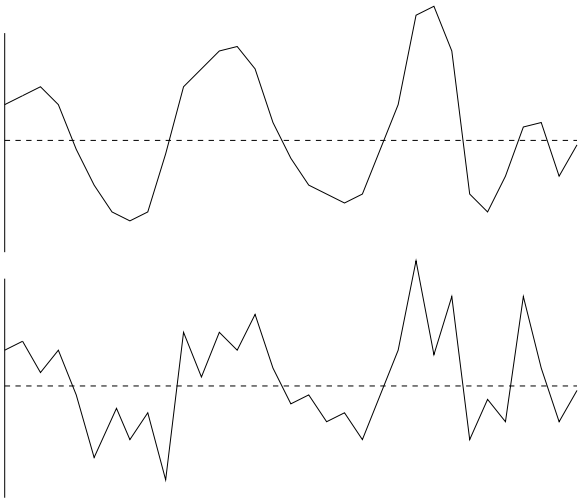
MINI-MAX: WRAPPING			
K	Gray Wins	Binary Wins	Ties
1	448	0	64
2	6752	2288	5552
3	6720	6592	9728
4	0	2160	16
Sum	13,920	11,040	15,360

Generating the Set of All Functions



Count the Minima in the Set of All Functions





A SubThreshold-Seeker

1. Evaluate a sample of points and estimate a $threshold(f)$.
2. Pick point $x < threshold(f)$.
3. If $f(x) < threshold(f)$ then set $x = x + 1$ and $y = x - 1$;
   Else sample a new random point.
4. While $f(x) < threshold(f)$ set $x = x + 1$;
5. While $f(y) < threshold(f)$ set $y = y - 1$;
6. If stopping-conditions not met, goto 2.

Define a *quasi-basin* as a contiguous set of points below threshold. Let α define a threshold presenting some fraction of the search space. Suppose there are B quasi-basins each containing at least M points.

Theorem: *Suppose that Subthreshold-Seeker is used to find B quasi-basins each containing at least M points. Forall $\alpha < 1/2$ subtheshold-seeker beats random search if $M > \sqrt{\frac{NH(B-1)}{B}}$.*

$\sqrt{\frac{NH(B-1)}{B}}$ does not reference α because M is derived from α.

What about a simple bit climber using Gray Code?

Theorem: *Given a quasi-basin that spans $1/Q$ of a search space of size N and a reference point R inside the quasi-basin, the expected number of neighbors of R that fall inside the quasi-basin under a reflected Gray code is greater than*

$$\lfloor (log(N/Q)) \rfloor - 1$$

Corollary: *Given a quasi-basin below theshold α that spans $1/Q$ of the search space and a reference point R that fall in the quasi-basin, the majority of the neighbors of R under a reflected Gray code representation of a search space of size N will also be subthreshold in expectation when*

$$\lfloor (log(N/Q)) \rfloor - 1 > \log(Q) + 1$$

This means that a simple "local search" bit climber can beat random enumeration when restarted from a subthreshold points as long as on average

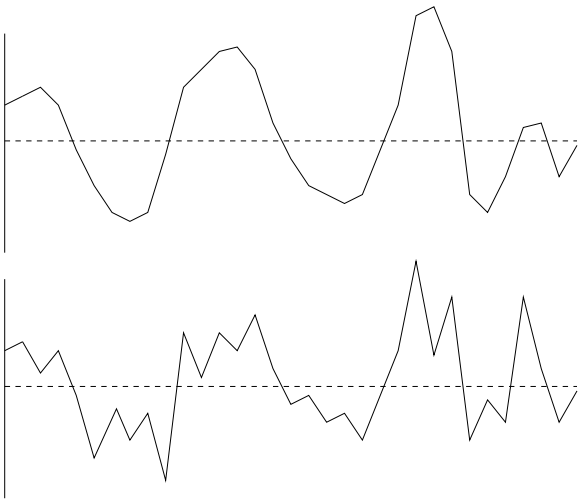
$$\lfloor(log(N/Q))\rfloor - 1 > \log(Q) + 1$$

Let $N = 2^{100}$ and assume we want to largely sample a quasi-basin that spans $1/billon^{th}$ of the space.

$$\lfloor(log(2^{100}/2^{30}))\rfloor - 1 > \log(2^{30}) + 1$$

$$69 > 31$$

NOTE: An increase in precision increases $\lfloor(log(N/Q))\rfloor - 1$ but does not increase $\log(Q) + 1$.



		10 bit Precision				*20 bit Precision*		
Func	ALG	Mean	Sub	Evals		Mean	Sub	Evals
ackley	R-LS	0.18	62.4	19371		0.0001	75.1	77835
	SubT	0.18	79.7	16214†		0.0001	89.9	73212†
grie-	R-LS	0.010	59.5	13412		0.0045	80.3	66609
wangk	SubT	0.005	80.1	9692†		0.0049	90.0	59935†
rana	R-LS	-49.6	49.5	22575		-49.76	74.2	3×10^6
	SubT	-49.4	57.6	19453†		-49.83	85.0	3×10^6

Table 1: Local Search Results averaged over 30 runs. Threshold = 10 percent. The † denotes statistical significance.

References

[1] S. Christensen and F. Oppacher. *What can we learn from No Free Lunch*. GECCO 2002.

[2] J. Culberson. On the Futility of Blind Search. *Evolutionary Computation*, 6(2):109–127, 1999.

[3] S. Droste and T. Jansen and I. Wegener. Perhaps not a free lunch but at least a free appetizer. *GECCO*, 1999.

[4] S. Droste and T. Jansen and I. Wegener. Optimization with randomized search heuristics; the (A)NFL theorem, realistic scenarios and difficult functions. *Theoretical Computer Science*, 2002 (In Press).

[5] T. English. Practical implications of new results in conversation of optimizer performance. *Parallel Problem Solving from Nature*, 2000.

[6] T. English. Information is Conserved in Optimization. *IEEE Trans Evolutionary Computation*.

[7] G. Rawlins. Introduction to "Fondations of Genetic Algorithms", 1991.

[8] N.J. Radcliffe and P.D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. *Lecture Notes in Computer Science 1000*. Springer-Verlag, 1995.

[9] C. Schumacher. *Fundamental Limitations of Search*. PhD thesis, University of Tennessee, Department of Computer Sciences, Knoxville, TN, 2000.

[10] C. Schumacher and M. Vose and D. Whitley. *The No Free Lunch and Problem Description Length*. *GECCO* 2001.

[11] D. Whitley. Functions as permutations: regarding no free lunch, walsh analysis and summary statistics. *Parallel Problem Solving from Nature, 6*, 2000.

[12] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

[13] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Trans Evolutionary Computation*, 4:67–82, 1997.