FOGA and THEORY: Past, Present, Future

Darrell Whitley
Computer Science
Colorado State University

In the mid-1990s there were interesting “breakthroughs” in the theory of Genetic Algorithms and Evolutionary Algorithms.

Exact Models ... “Corrections” to Holland’s theories and conjectures.
Theory (and FOGA) was hot.

In the last 10 years,
there is decreased interest in theory from the general EC community.

o

Sometimes Theory is extremely useful.

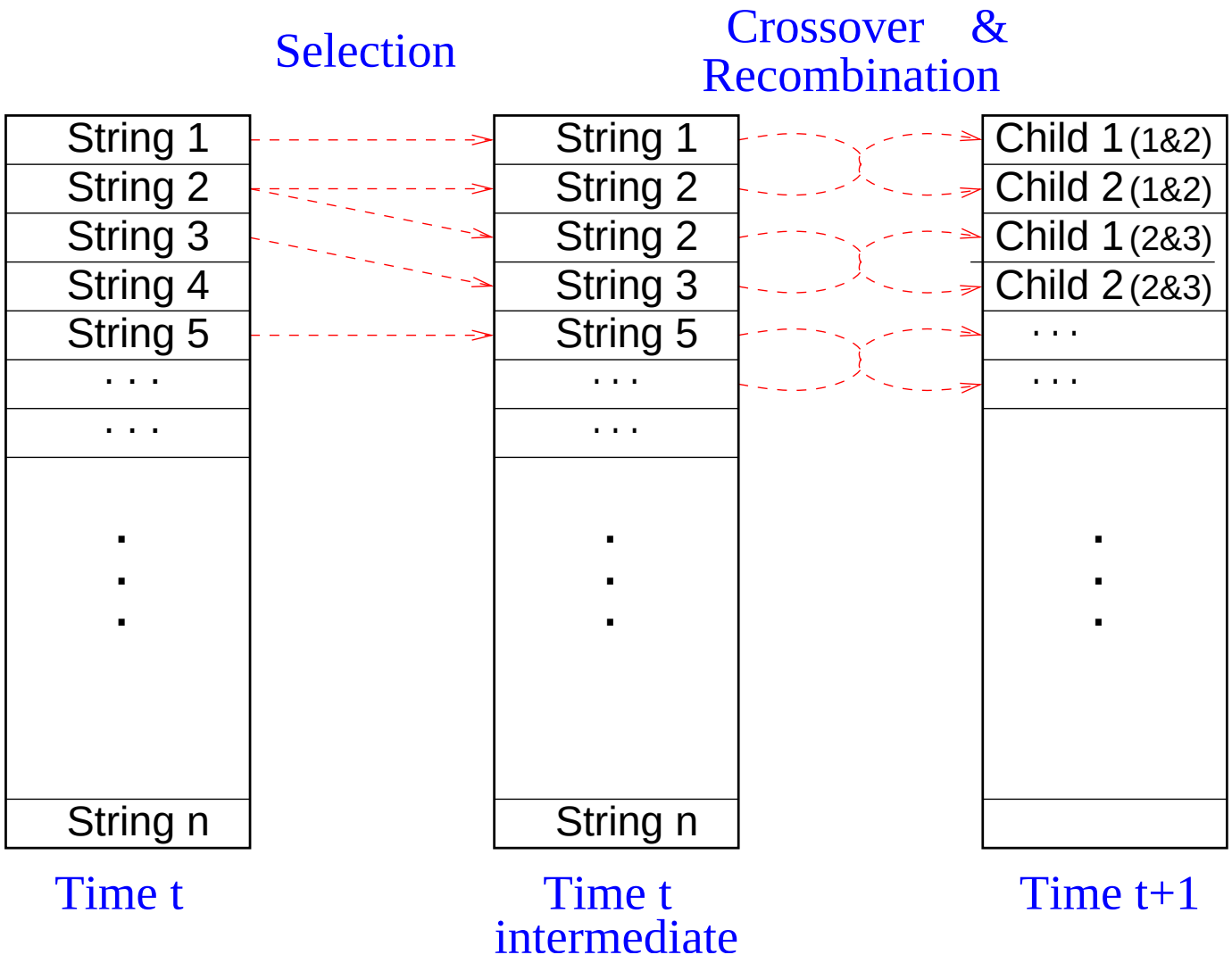
But sometimes Theory is like caffeine free diet coke:

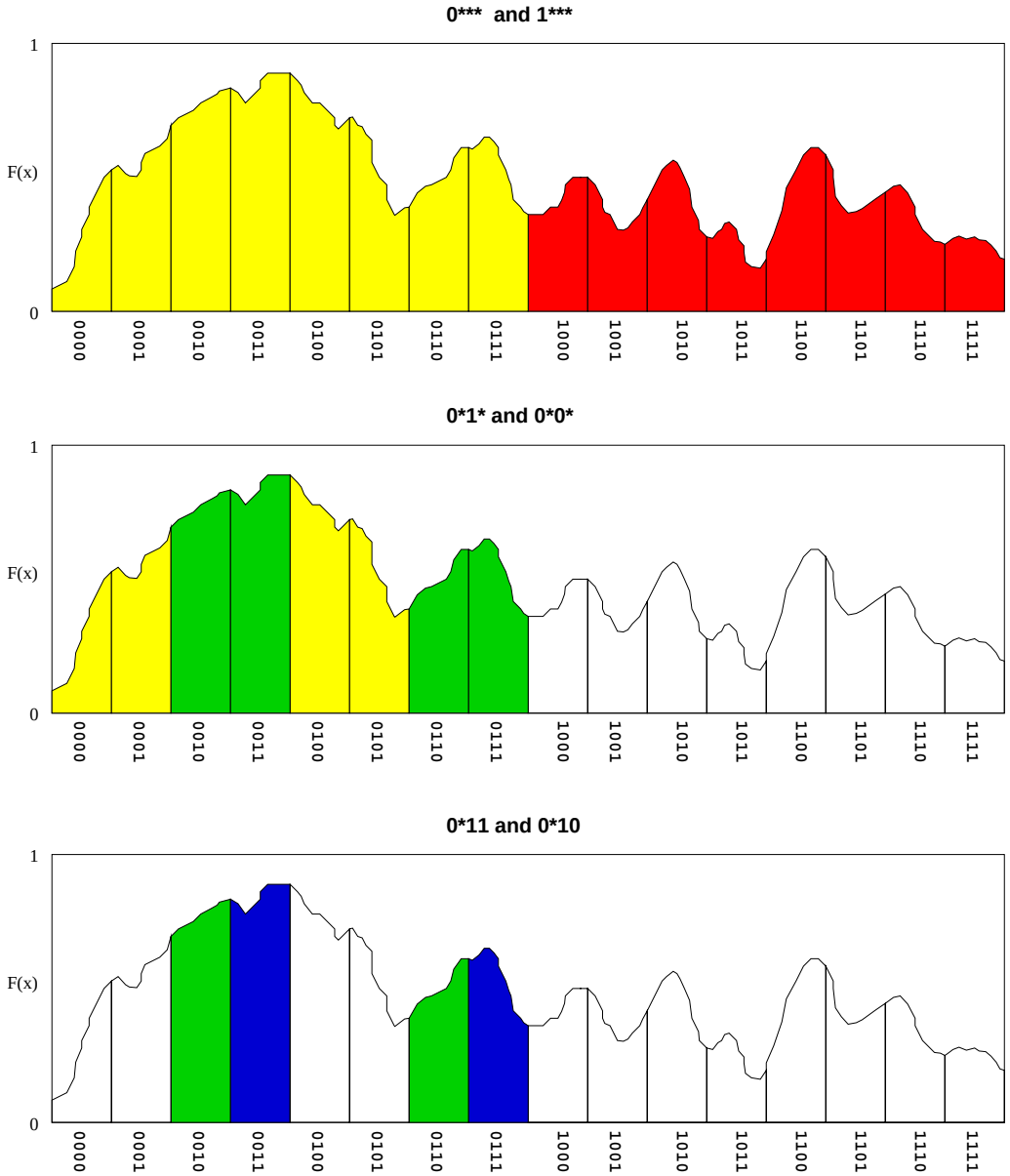
Sure, its fun, but what's the point?

What is the role of theory?

What are some alternative approaches to theory?

SIMPLE GENETIC ALGORITHM MODEL




0*** and 1***
1
F(x)
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0*1* and 0*0*
1
F(x)
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0*11 and 0*10
1
F(x)
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

THE SCHEMA THEOREM and EXACT MODELS

Selection Only: $P(H, t + intermediate) = P(H, t)\frac{f(H)}{\bar{f}}$.

An Exact Calculation:

$$P(H, t+1) = P(H, t)\frac{f(H)}{\bar{f}}(1 - p_c\, losses) + p_c gains$$

A Common Version of the "Schema Theorem":

$$P(H, t+1) \geq P(H, t)\,\frac{f(H)}{\bar{f}}\,\left[1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H, t)\,\frac{f(H)}{\bar{f}})\right]$$

THE SCHEMA THEOREM and EXACT MODELS

An Exact Calculation:

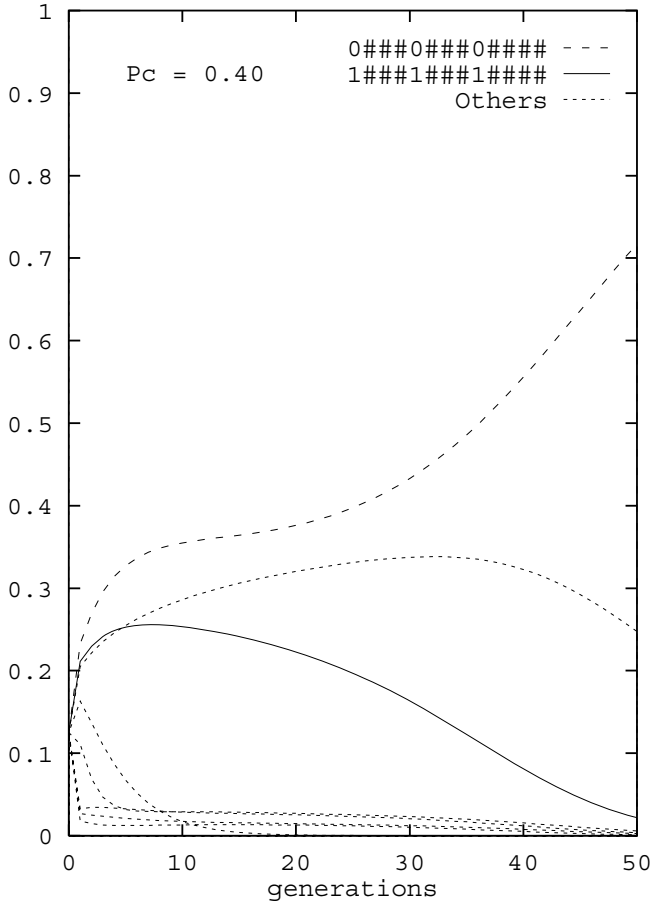
$$P(i,t+1) = P(i,t)\frac{f(i)}{\bar{f}}(1 - p_c\ losses) + p_c gains$$

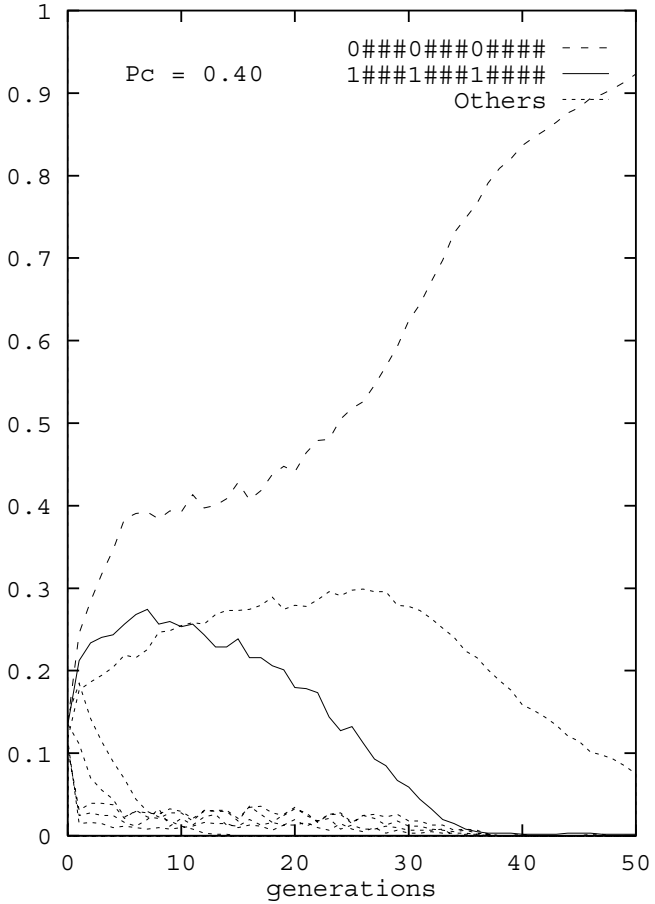
The Vose/Liepins Model:

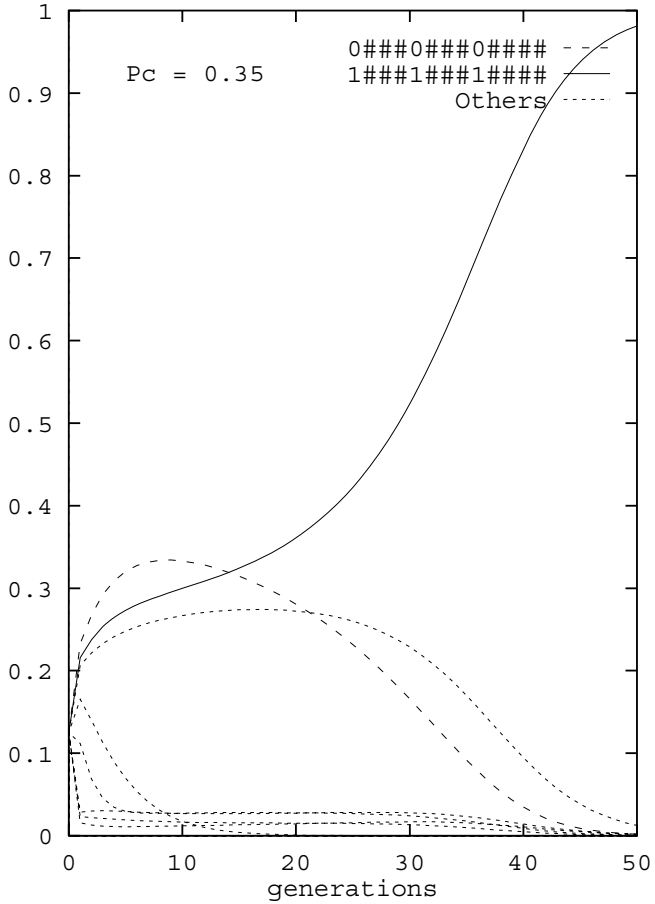
$$s_i^t = P(i,t)f(i)/\bar{f}$$

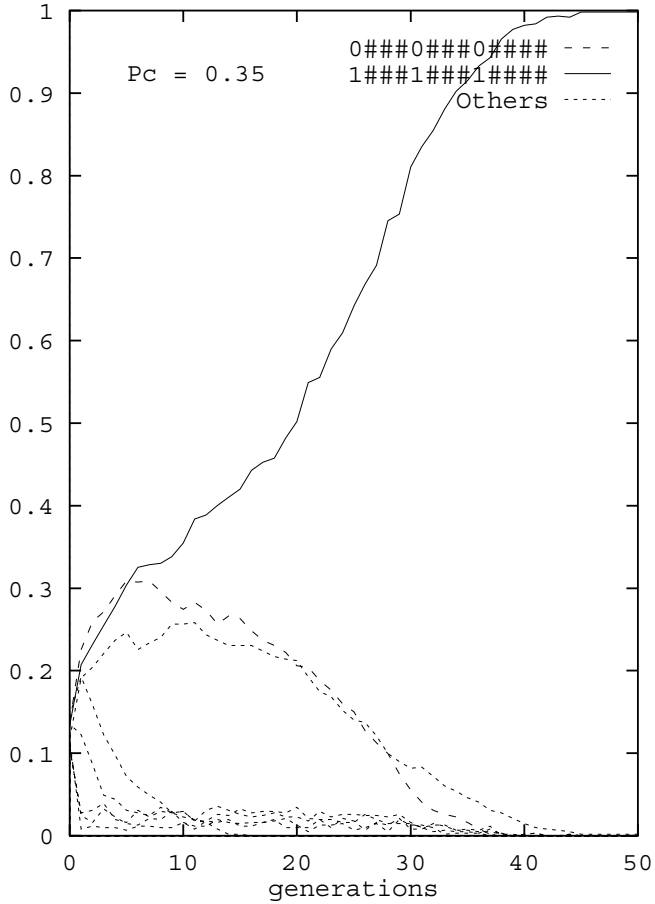
$$p_i^{t+1} = (\rho_i\ s)^T M(\rho_i\ s) = P(i,t)\frac{f(i)}{\bar{f}}(1 - p_c\ losses) + p_c gains$$

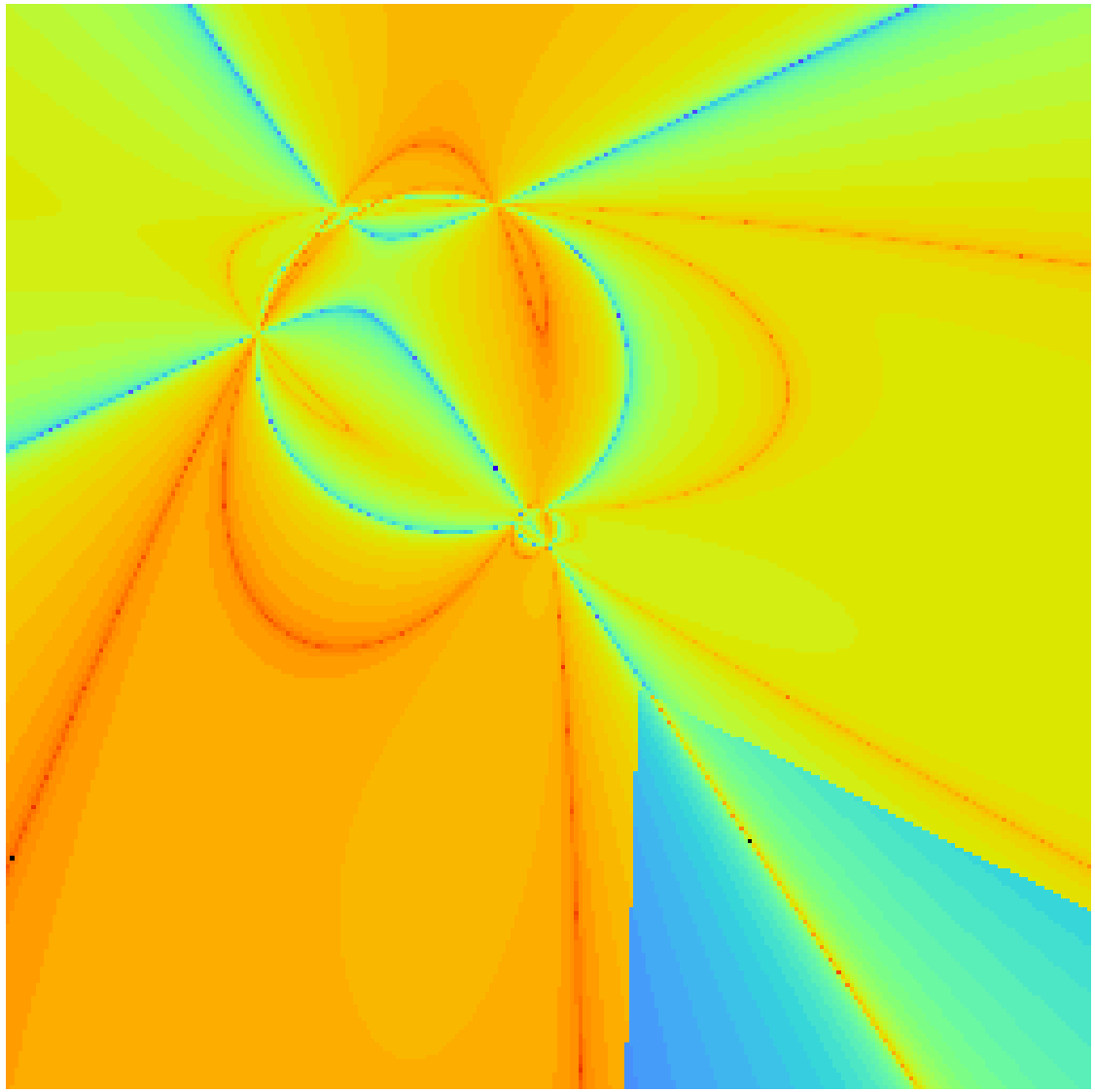
The function $r_{i,j}(k)$ is used to construct a mixing matrix M where the i,jth entry $m_{i,j} = r_{i,j}(0)$. This matrix gives the probabilities that crossing strings i and j will produce S_0.

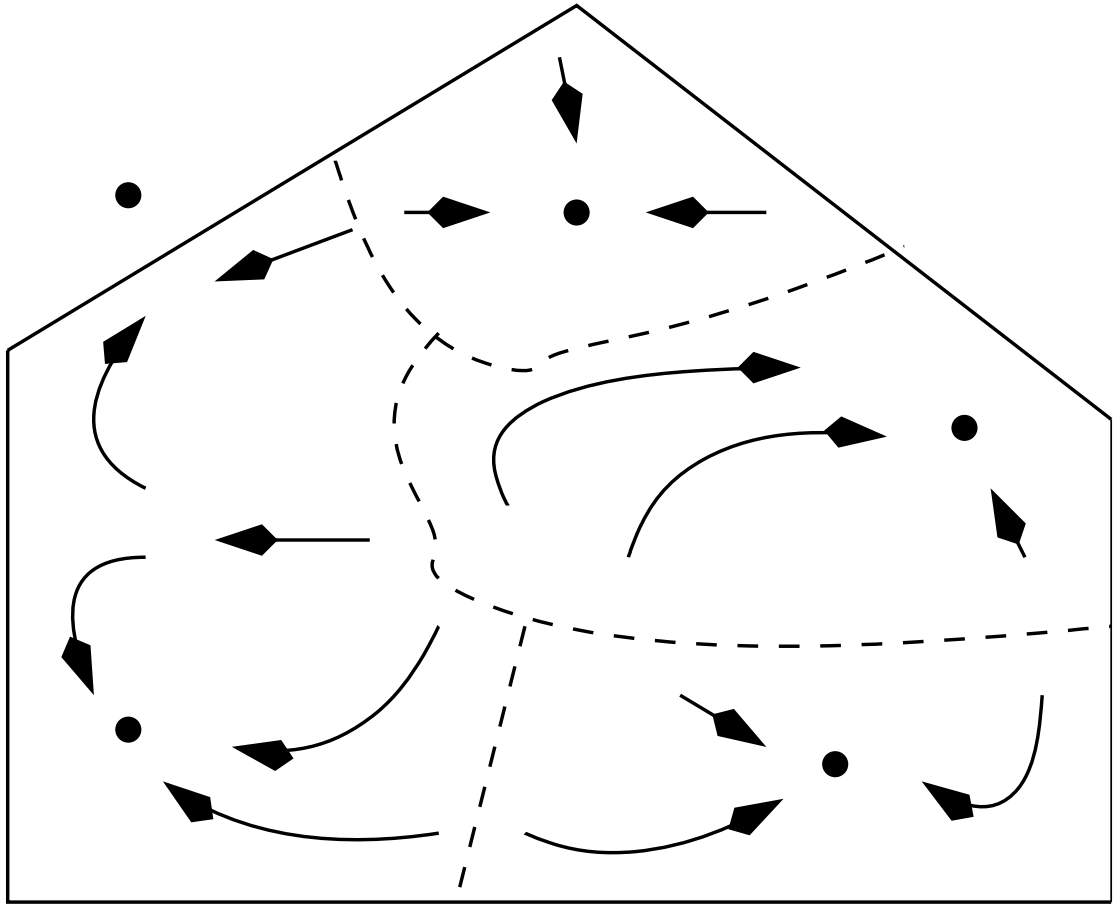

Pc = 0.40
0###0###0####
1###1###1####
Others
generations


Pc = 0.40
0###0###0####
1###1###1####
Others
generations


Pc = 0.35
0###0###0####
1###1###1####
Others
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
generations


Pc = 0.35
0###0###0####
1###1###1####
Others
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
generations





Three Problems with Evolutionary Algorithm Theory

Problem 1: Theory versus Practice

The theory does not apply to algorithms that are widely used.

Problem 2: Theory versus Theorems

The theory community deals almost exclusive with theorems as opposed to developing explanatory ”theory.”

Problem 3: Toy and Artificial Test Problems

Does theory (or empirical results) for a toy problem tell us anything meaningful about real world problems?

Problem 1: Theory versus Practice

- **Simple Genetic Algorithm**
- **Evolution Strategies**
- **CMA Evolution Strategies**
- **Genetic Programming**
- **Genitor, Steady-State GAs**
- **CHC**
- **Memetic Algorithms**
- **Parallel Genetic Algorithms**

Theory does not cover the most widely used methods.

In practice, algorithms are customized to fit the problem.

```
Unique  ...  Specific ...  General ...  Black Box
             Classes       Classes     Optimization
<-------------------------------------------------->
```

Do we tie our hands my only considering black box optimization?

No Free Lunch suggests we cannot win at black box optimization.

Problem 2: “Theorems” versus “Theory”

In Physics, theoretical models are developed to explain physical phenomena. A “Theory” suggests testable hypotheses.
Our theory is almost all “theorem” based.

Even if we do not develop an overall “theory,” our theoretical (and empirical) work should embrace more “hypothesis testing.”

Problem 3: Toy and Artificial Test Problems

This is a problem for both Theory and Empirical Research.

1. Test Functions can be TOO EASY
   E.G. ONEMAX, Sphere Functions

2. Test Functions can be TOO HARD
   E.G. Random Job Shop Problems, N-K Landscapes

3. Test Functions can be UNREALISTIC
   E.G. Deceptive Functions and Trap Functions
   (These have theoretical value, but ....)

4. Test Functions can be TOO SPECIALIZED
   E.G. MAXSAT: Too many flat plateaus, specialized data structures for truth value of clauses.

It is *true* that there are no easy answers.

What else can we do? (Or encourage.)

3 examples of research mixing theory and practice.

1. Covariance Matrix Adaptation
   A *Constructive* Approach to Theory.

2. Using Markov Models to Model *Search Cost*.
   –Tabu Search and Job Shop Scheduling
   –A Genetic Algorithm for the Air Force Satellite Control Network

3. No Free Lunch and SubThreshold Seeking

CMA Covariance Matrix Adaptation *Nikolaus Hansen et al.*

A more constructive approach to algorithm design.

Let $\mathbf{Z}^{(g+1)}$ be the covariance of the μ best individuals.

Let $\mathbf{P}^{(g+1)}$ be the covariance of the evolution path.

The new covariance matrix:

$$\mathbf{C}^{(g+1)} = (1 - c_{cov})\mathbf{C}^{(g)} + c_{cov}\left(\alpha_{cov}\mathbf{P}^{(g+1)} + (1 - \alpha_{cov})\mathbf{Z}^{(g+1)}\right)$$

Where c_{cov} and α_{cov} are constants that weight each input.

CMA Covariance Matrix Adaptation

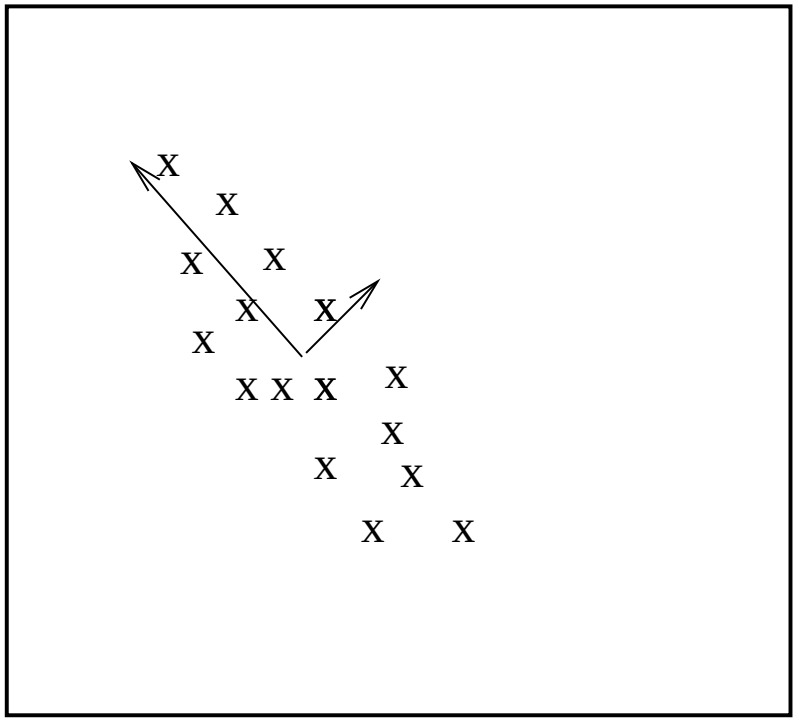
Let $\mathbf{Z}^{(g+1)}$ be the covariance of the μ best individuals (**Z**=Sample).

Note we sample λ points and keep the μ best. The default is $\mu = \lambda/2$.

Let $\mathbf{P}^{(g+1)}$ be the covariance of the evolution path (**P**=Path).

In effect, these covariance matrices are using Principal Component Analysis to determine the direction of maximum variance in both the Sample and the Path.

NOTE: we will not *really* use the Sample and Path, but rather direction of movement in the Sample and Path.



CMA and Principal Components

Given a data set of sample points, we want to perform an eigenvalue/eigenvector decomposition. The eigenvectors are represented by a rotation matrix **R**. Let Λ be the diagonal eigenvalue matrix. Let **X** represent a matrix of data vectors. Using PCA we find **R** and Λ such that

$$\mathbf{R} \cdot \mathbf{X}\mathbf{X}^T = \Lambda\mathbf{R}$$

CMA and Principal Components

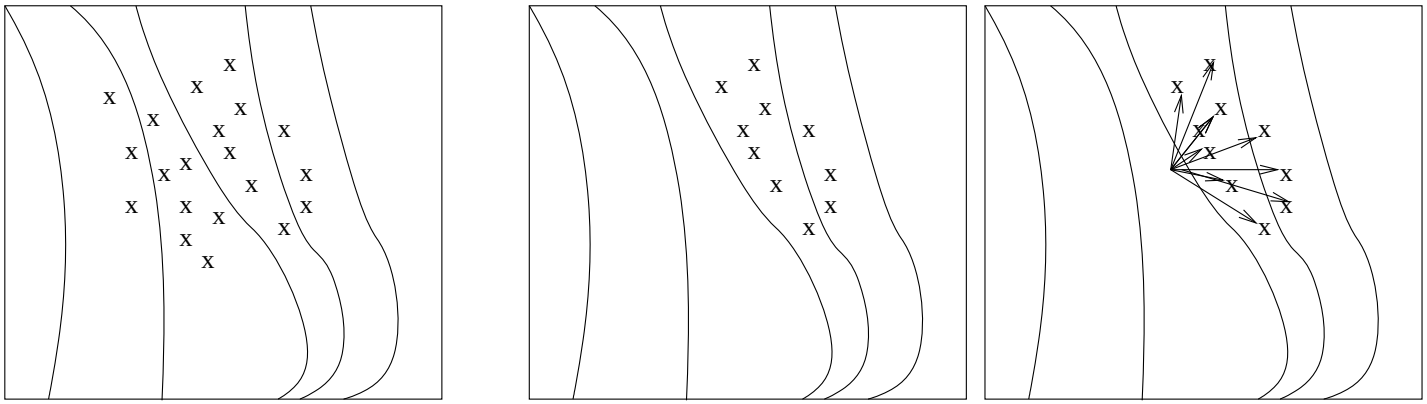
Why do we need a sample over the best **Offspring** *and* over the **Path**?

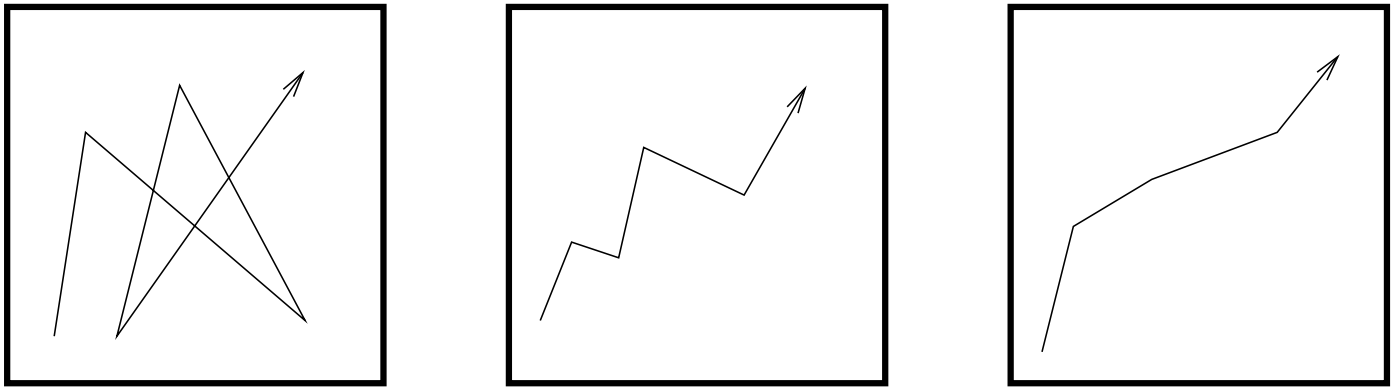
The direction of maximum variance over the best offspring may follow the gradient, or be orthogonal to the gradient and path.

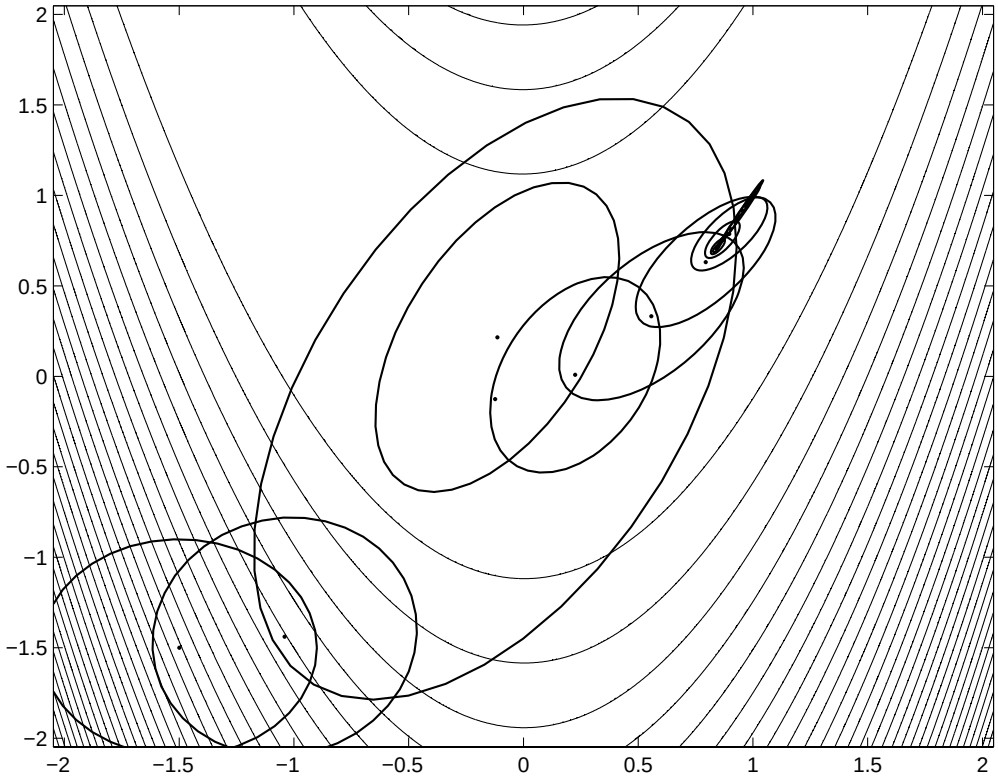
One can partly correct this by using the **direction** from the centroid of the parents to the offspring.

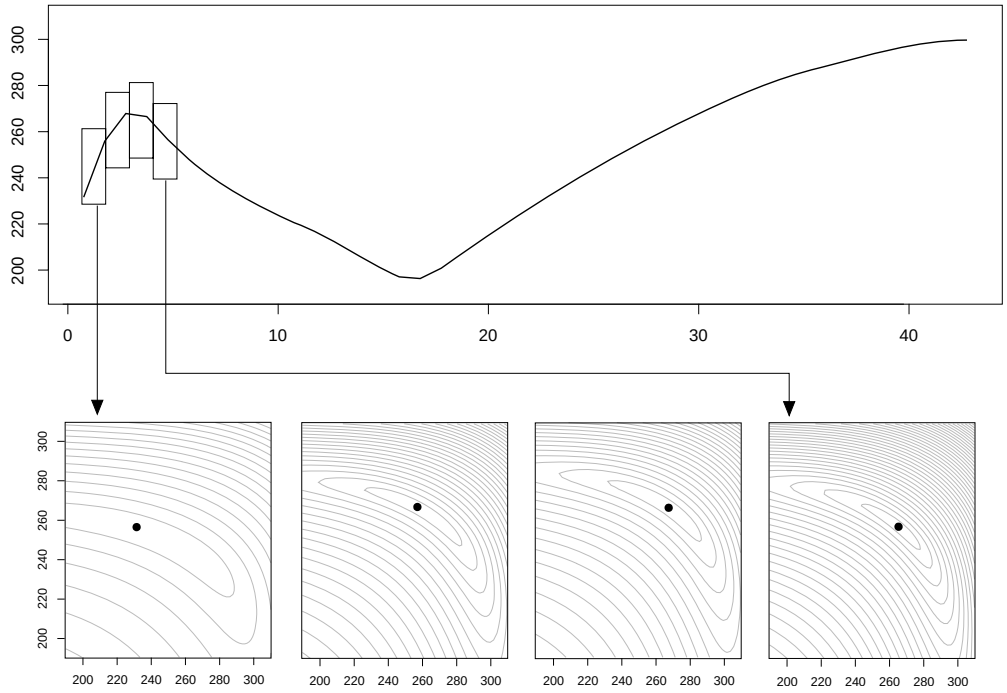
The direction of maximum variance over the path helps.

A heuristic related to path length also helps to determine step size.





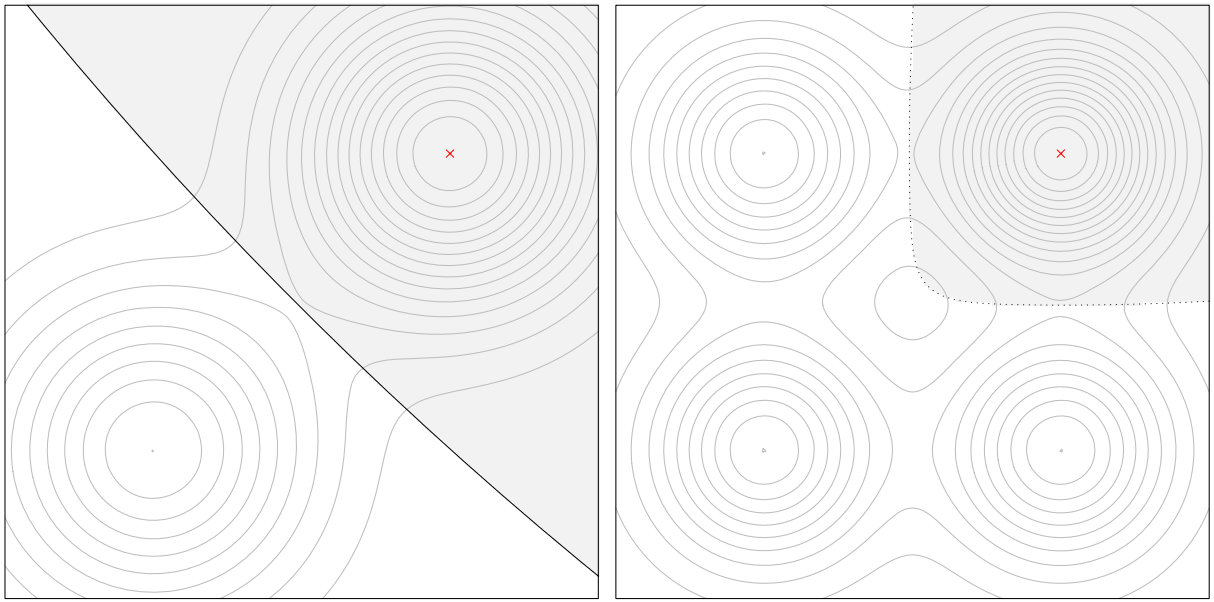

2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2

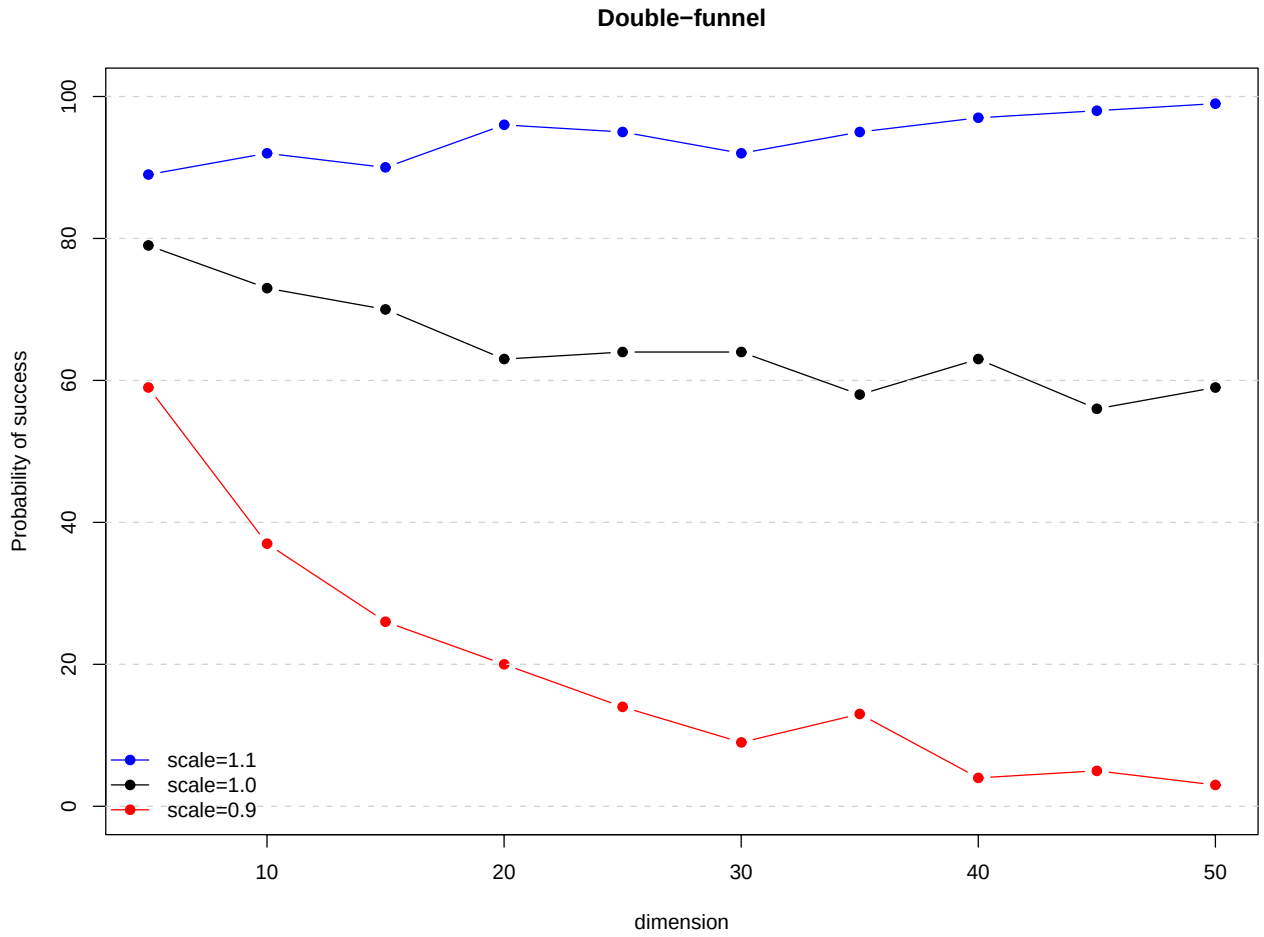

200
220
240
260
280
300
0
10
20
30
40
200 220 240 260 280 300

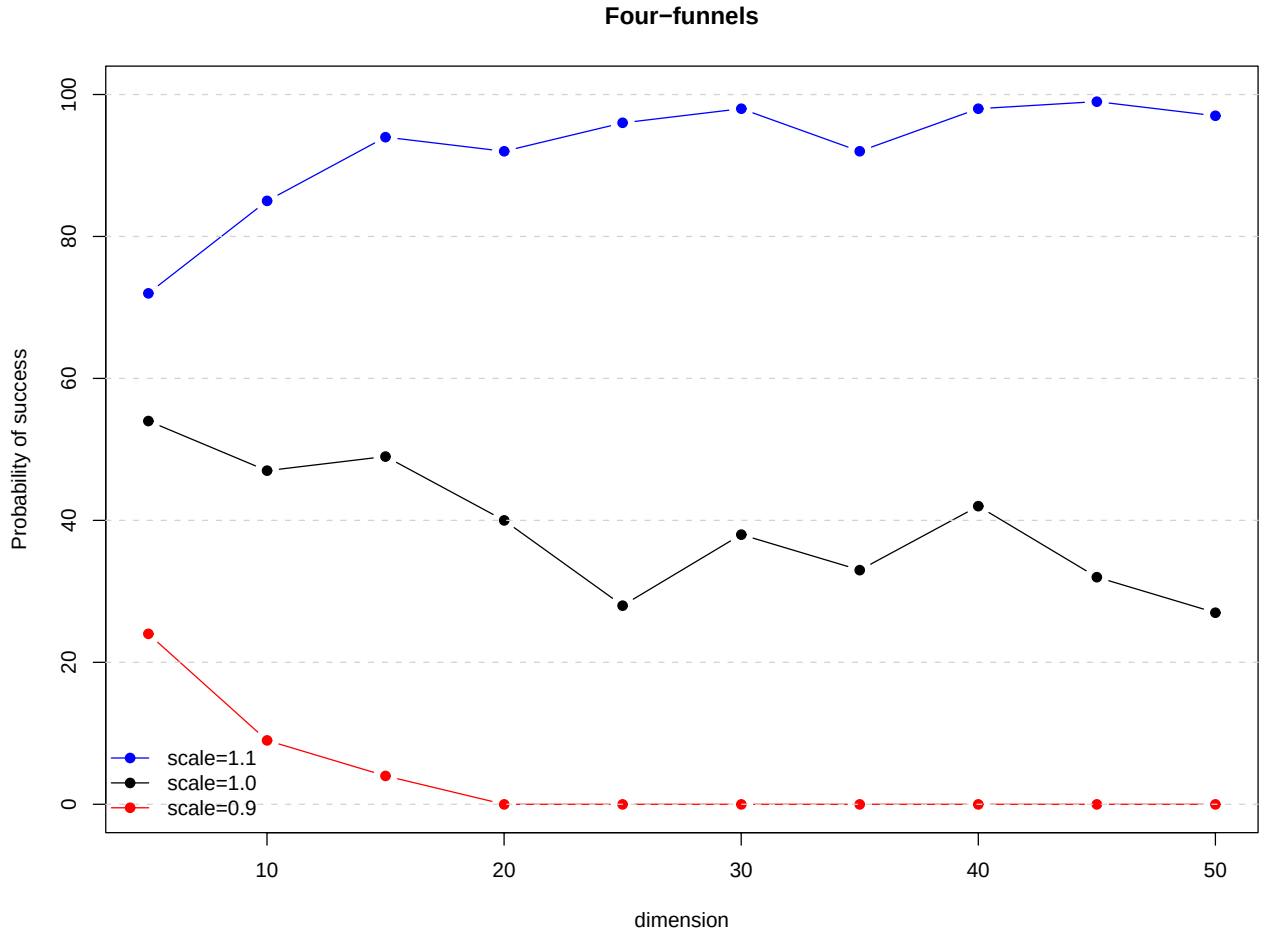
The Bias in CMA:

A very strong bias in CMA-ES is the (implicit) assumption that the function is a bowl (sphere or elipse) or has a global bowl-like (funnel) structure.

So what if we construct very simple functions with 2 bowls or 4 bowls?




Double−funnel
Probability of success
100
80
60
40
20
0
10
20
30
40
50
dimension
scale=1.1
scale=1.0
scale=0.9


Four−funnels
Probability of success
0
20
40
60
80
100
10
20
30
40
50
dimension
scale=1.1
scale=1.0
scale=0.9

The $n\ x\ m$ Job-Shop Scheduling Problem

For the JSP n jobs must be processed exactly once on each of m machines.

Each job i is routed through the m machines in some pre-defined order π_i, where $\pi_i(j)$ denotes the jth machine in the routing order.

Most research considers the problem of makespan minimization.

TABU Search has been the best method to solve JSP for 15 years.

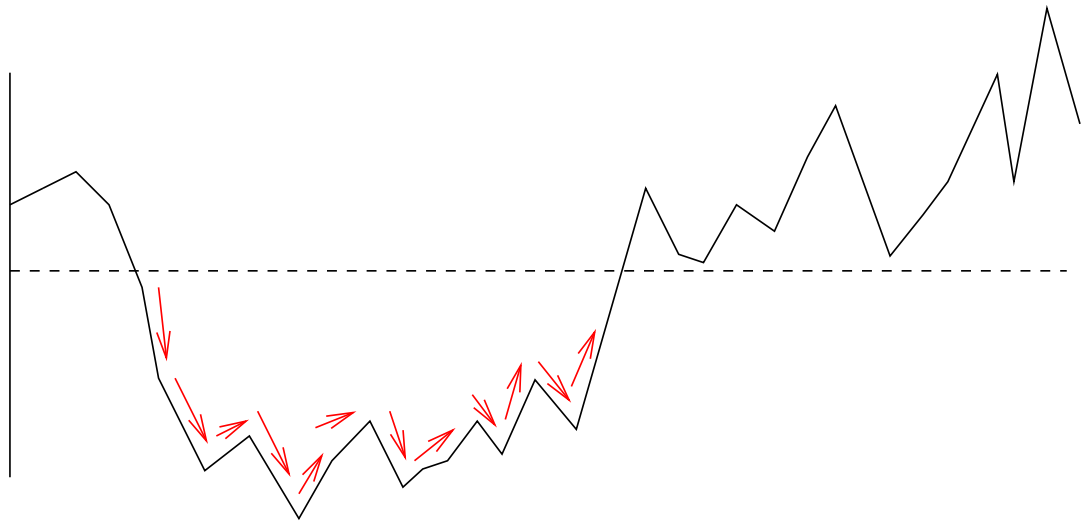
Cost Models for Job Shop Scheduling: *Jean Paul Watson*

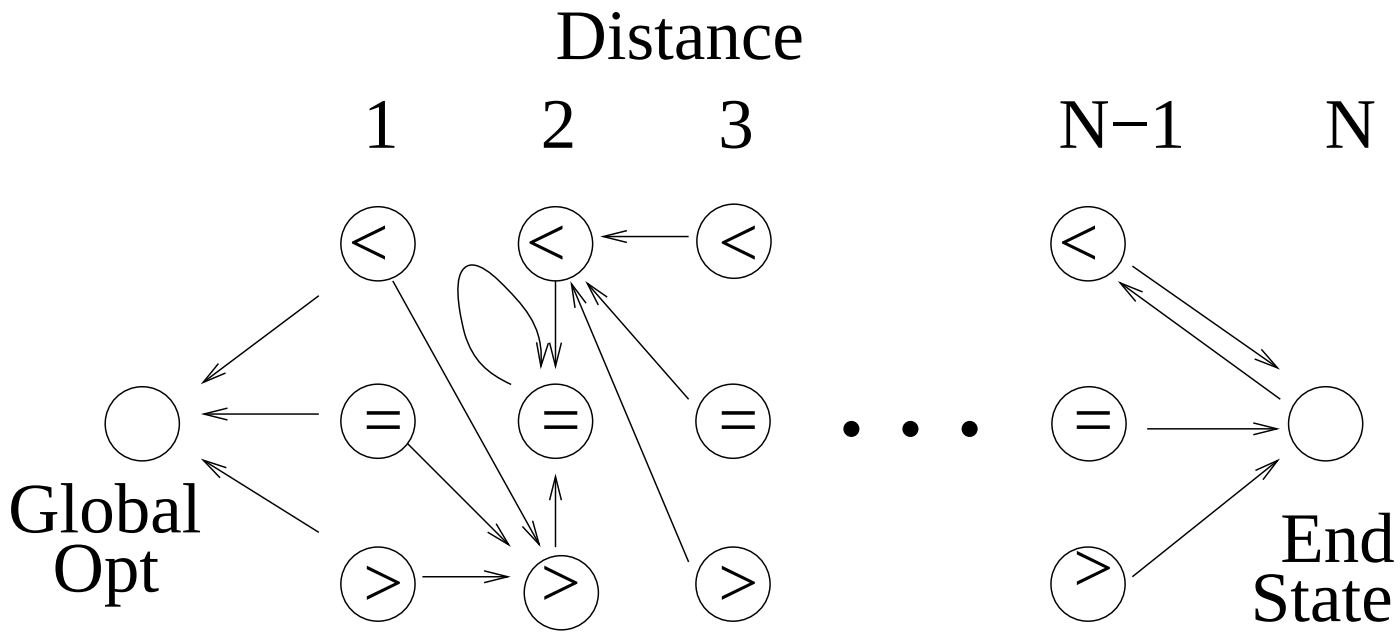
What accounts for search cost:

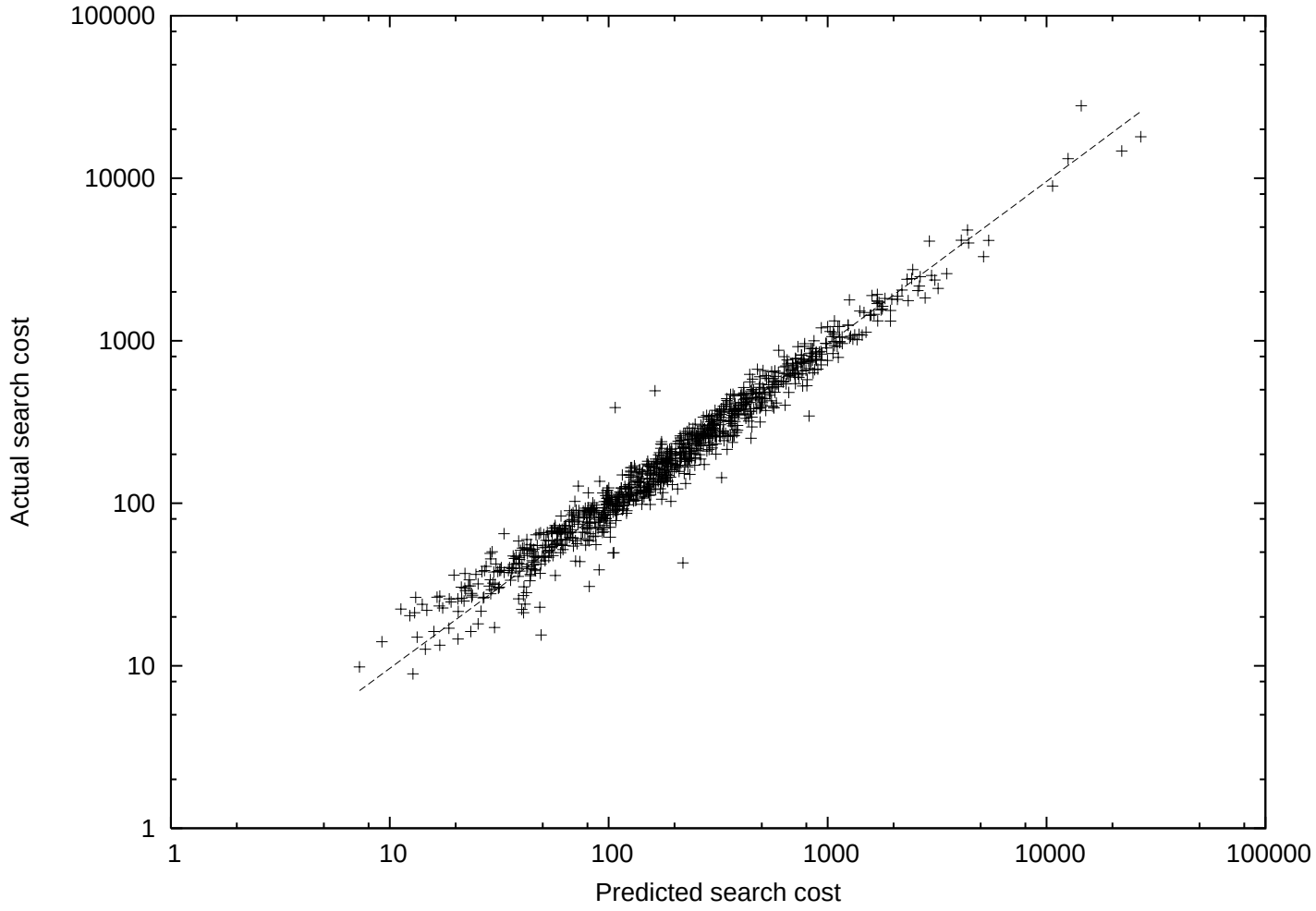
1. number of local optima?
2. fitness distance correlation?
3. backbone size?
4. distance to the global optimum? ...

After extensive study, it seems that distance to the global optimum combined with a thresholding of the local optima were most predictive of search cost.

Plus there was a momentum effective from the Tabu List.

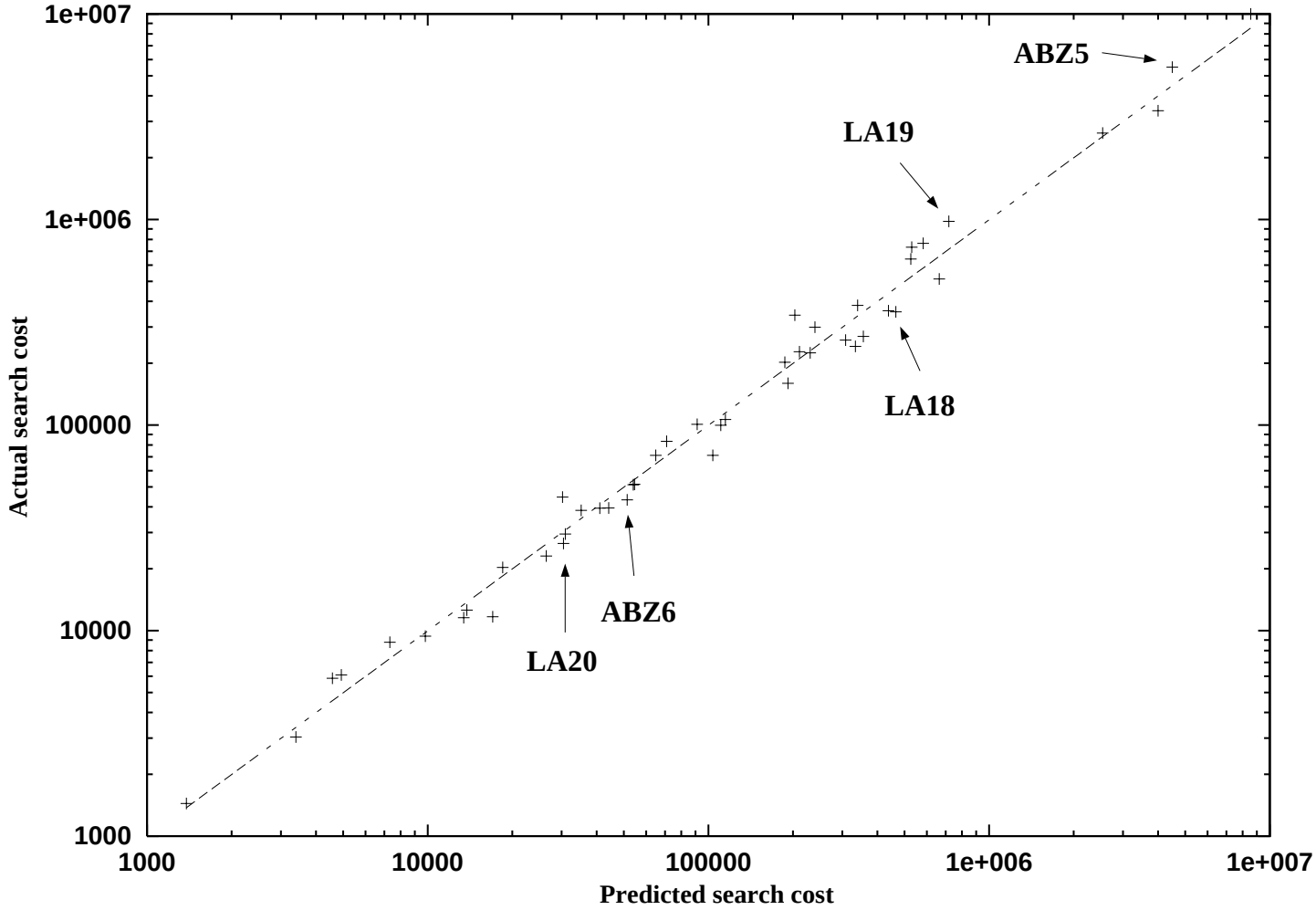



Distance
1
2
3
N−1
N
Global
Opt
End
State



Predicted Search Costs for 6X6 job shop problem benchmarks.

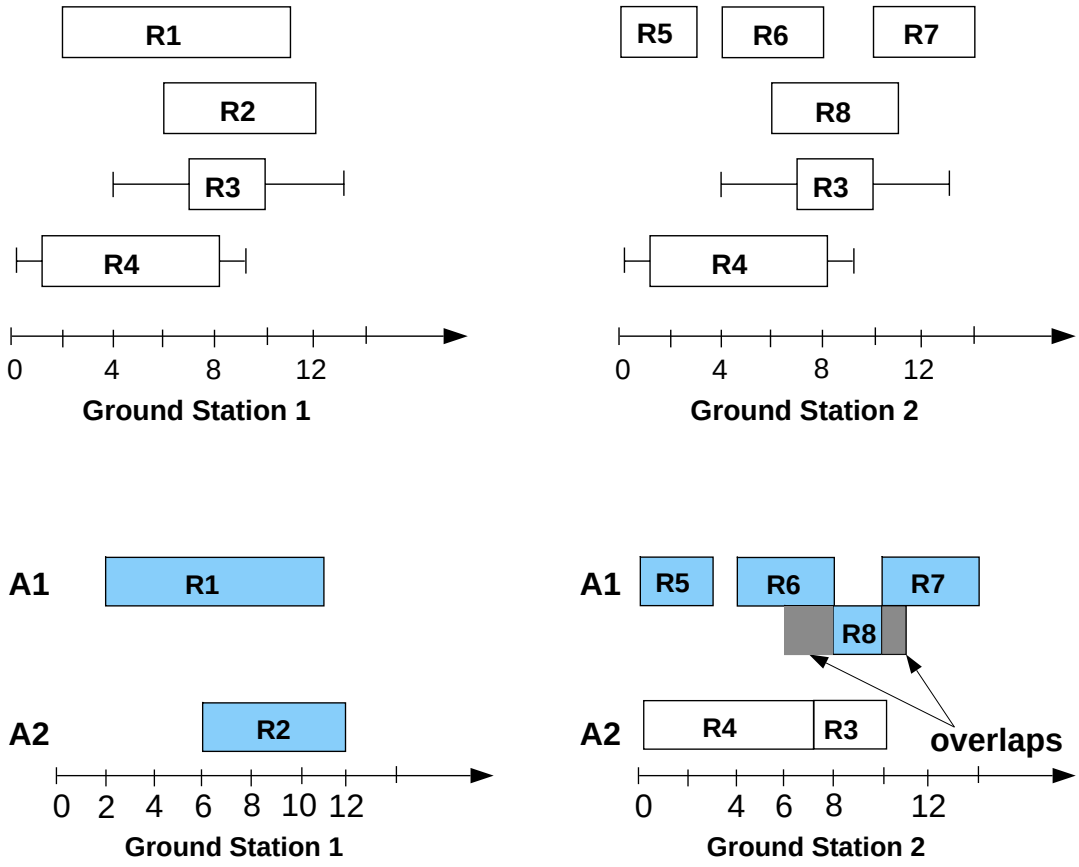
R^2 value = 0.96



Predicted Search Costs for 10X10 job shop problem benchmarks.

These models allowed us to construct a simple local search method that beats TABU Search.

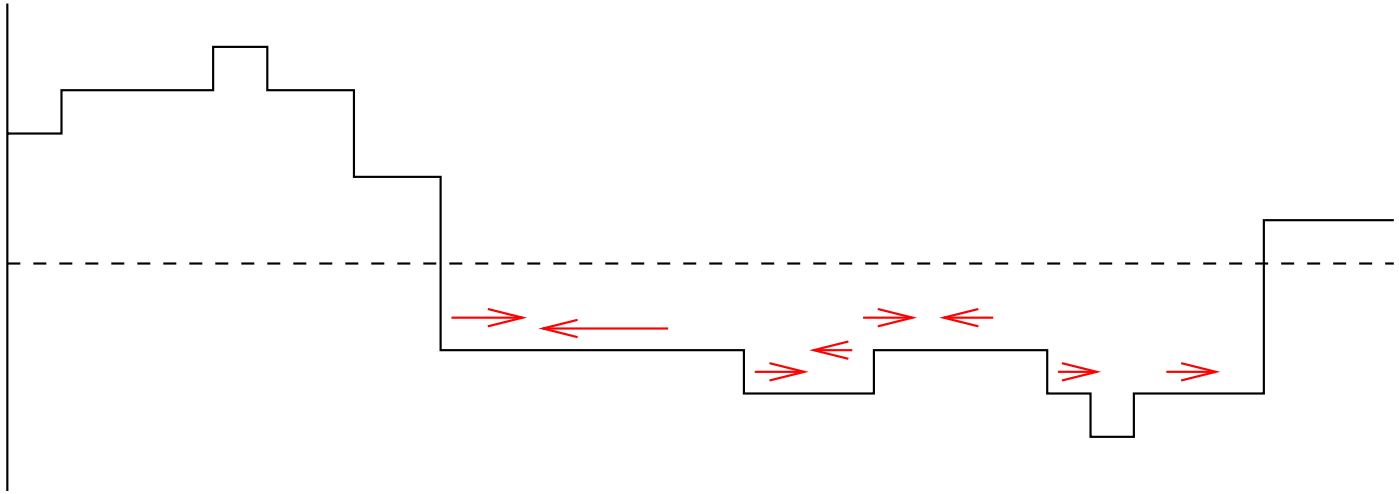
The Air Force Satellite Control Network processes

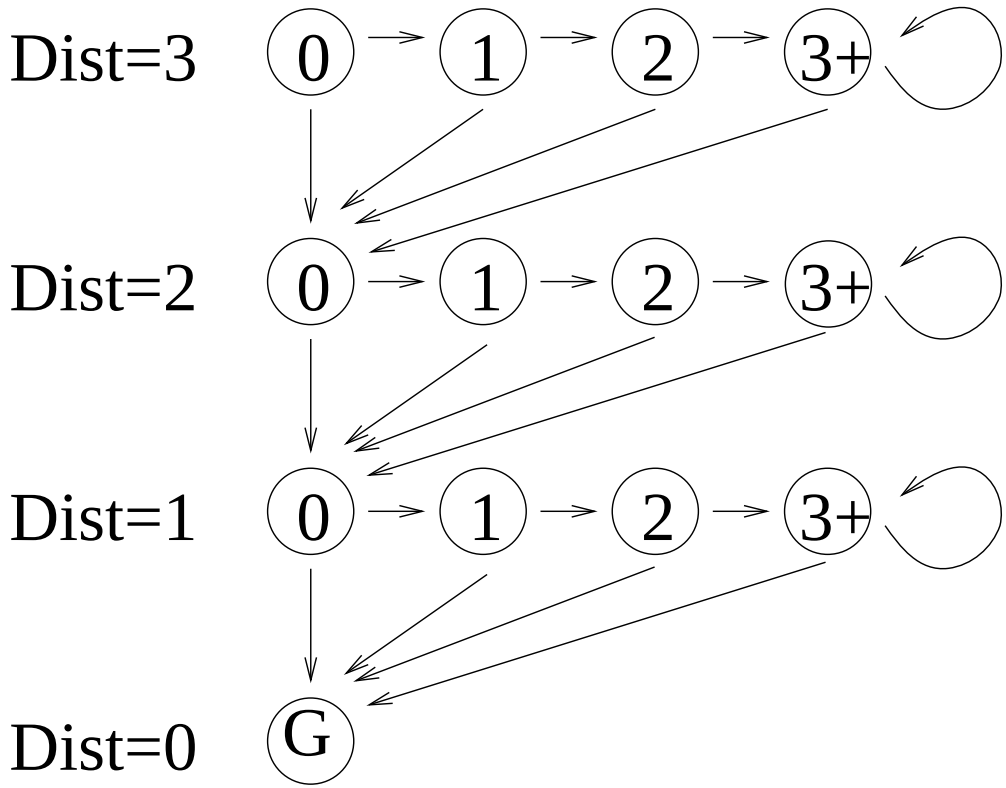


Approximately 500 contacts (requests) a day are scheduled
using 16 groundstations, 2 antemna per station.

A steady state Genetics Algorithm was the best known scheduling method.

How did the GA work?




Dist=3
0
1
2
3+
Dist=2
0
1
2
3+
Dist=1
0
1
2
3+
Dist=0
G

These models allowed us to construct a simple local search method that sometimes beats the GA.

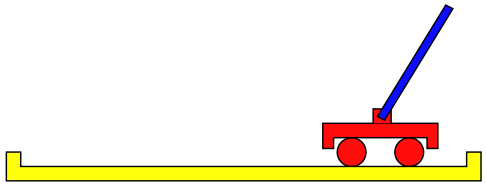
Schema Processing vs Hill-Climbing for Evolving Programs

- For Hill-Climbing, we used a **(1,10)-ES** and **(1+10)-ES**.
- Mutation:

  A subtree is selected and randomly regenerated using ”Grow.”
- We used Sean Luke’s ECJ Implementation.
- All experiments were run for 100,000 evaluations.
- We used 20 runs for each experiment for each problem.

Test Problems

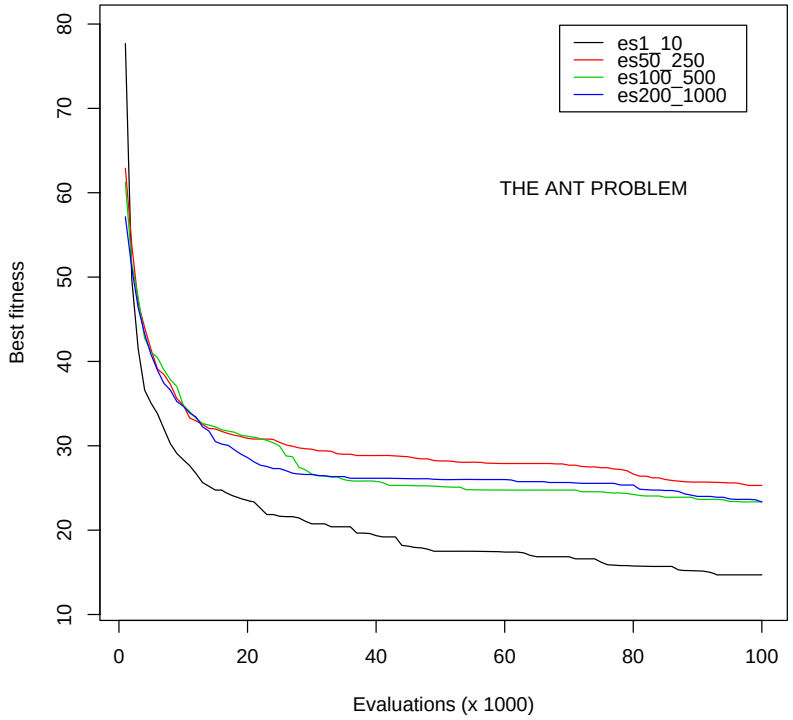
- Artificial Ant eating food in a 2-D world.
- 11-Multiplexor: a1 a2 a3 b0 b1 b2 b3 b4 b5 b6 b7
- Symbolic Regression: $x^6 - x^4 + x^2$ sampled from -1 to 1.
- Cart and Pole Balancing

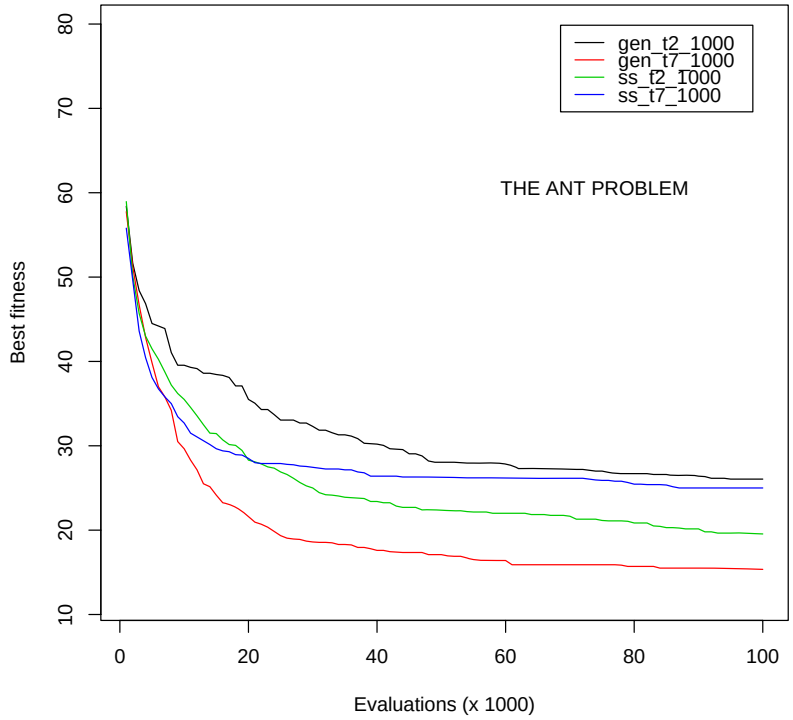


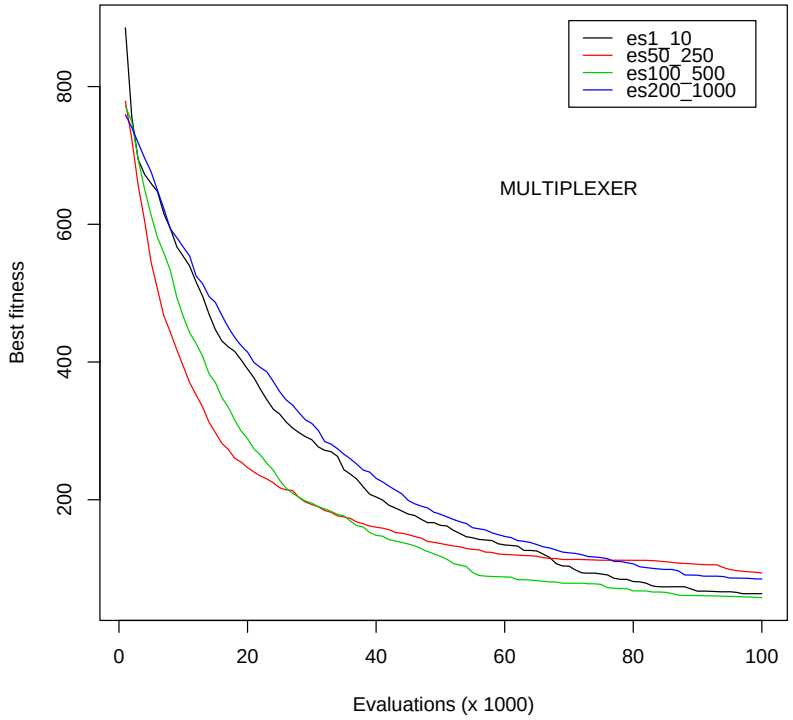
Comparative Trends

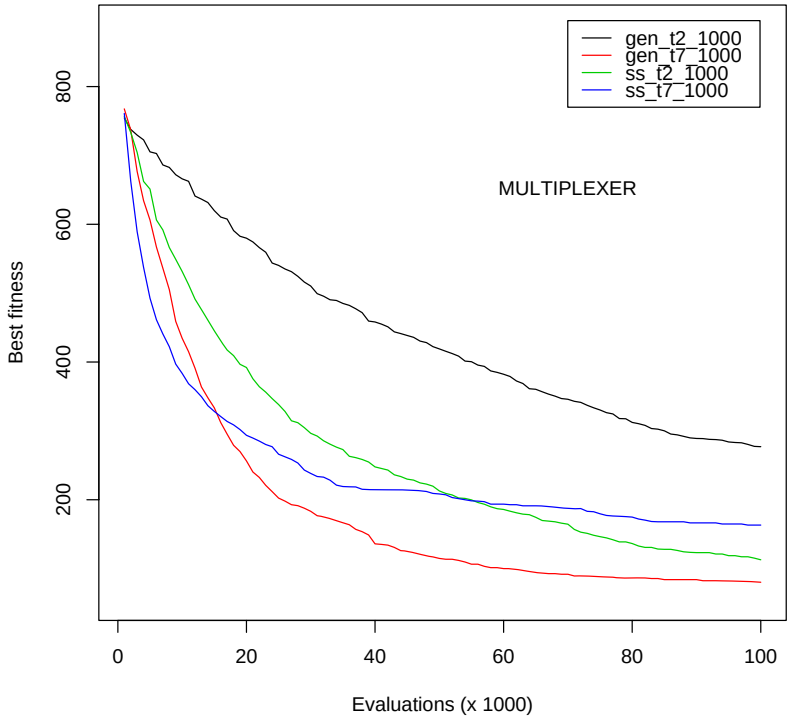
	ant	**multi11**	**symb**	**pole**
(1,10)-ES	14.70	63.60	0.039	193.05
(200,1000)-ES	23.35	85.00	0.052	256.40
(1+10)-ES	21.55	69.60	0.156	276.40
(200+1000)-ES	12.00	79.20	0.034	227.60
Steady State GP t2 1000	15.25	111.70	0.049	214.85
Generational GP t7 1000	16.30	72.00	0.058	249.25

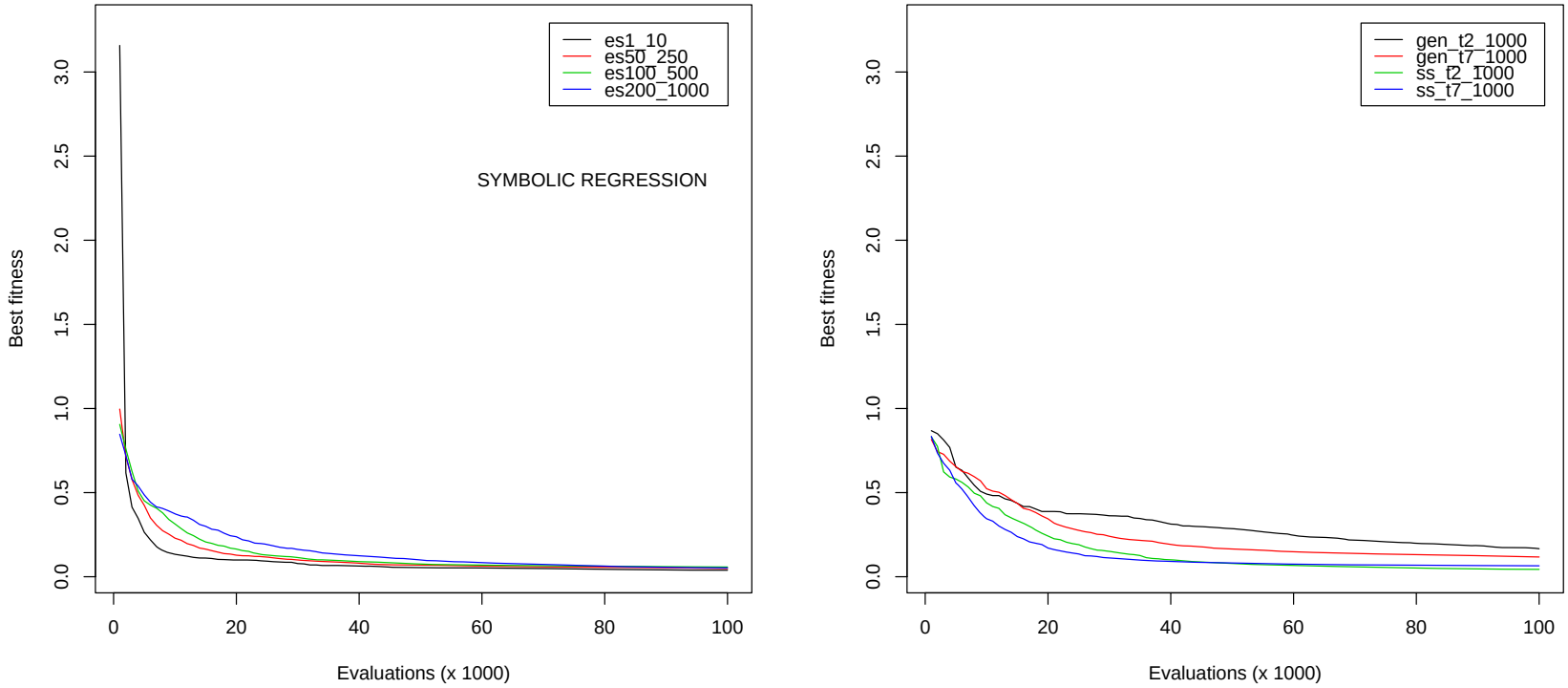
- The (1,10)-ES *dominates* the two GP methods.
- The (1,10)-ES is a stochastic hill-climber that accepts non-improving moves.

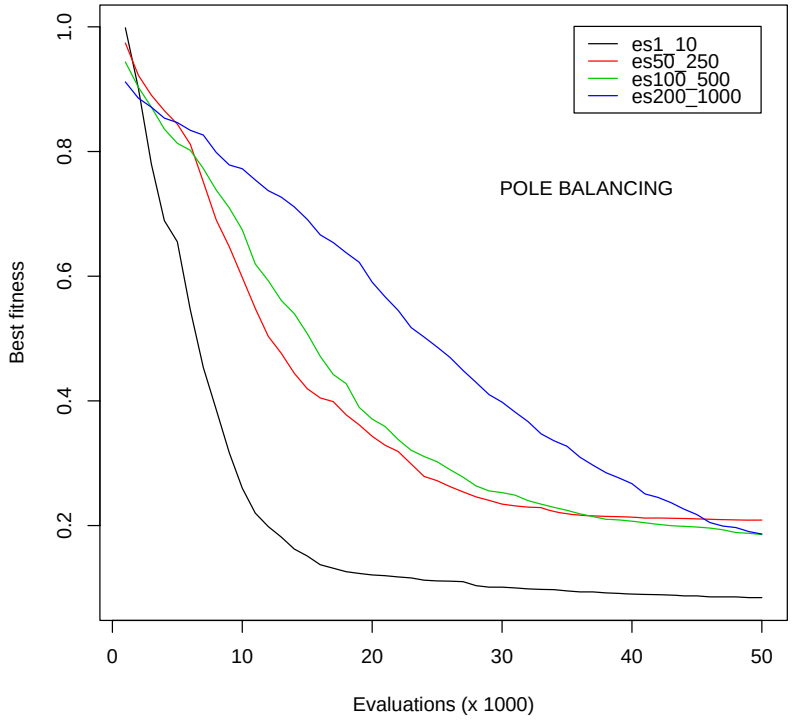

es1_10
es50_250
es100_500
es200_1000
THE ANT PROBLEM
Best fitness
Evaluations (x 1000)
10
20
30
40
50
60
70
80
0
20
40
60
80
100

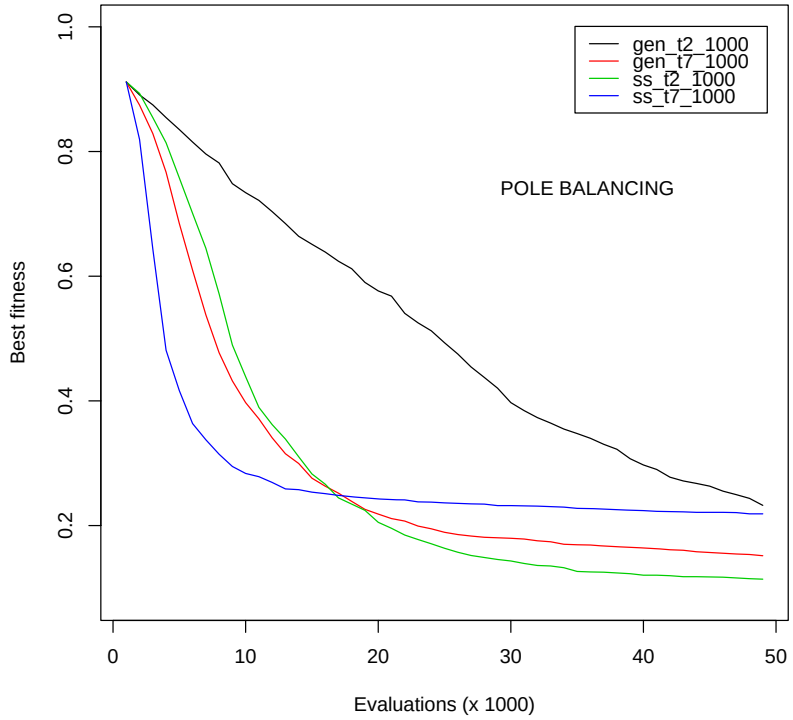

gen_t2_1000
gen_t7_1000
ss_t2_1000
ss_t7_1000
THE ANT PROBLEM
Best fitness
Evaluations (x 1000)
10
20
30
40
50
60
70
80
0
20
40
60
80
100


es1_10
es50_250
es100_500
es200_1000
MULTIPLEXER
Best fitness
200
400
600
800
0
20
40
60
80
100
Evaluations (x 1000)


gen_t2_1000
gen_t7_1000
ss_t2_1000
ss_t7_1000
MULTIPLEXER
Best fitness
200
400
600
800
0
20
40
60
80
100
Evaluations (x 1000)


es1_10
es50_250
es100_500
es200_1000
SYMBOLIC REGRESSION
Best fitness
Evaluations (x 1000)
gen_t2_1000
gen_t7_1000
ss_t2_1000
ss_t7_1000
Best fitness
Evaluations (x 1000)


es1_10
es50_250
es100_500
es200_1000
POLE BALANCING
Best fitness
Evaluations (x 1000)
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50


gen_t2_1000
gen_t7_1000
ss_t2_1000
ss_t7_1000
POLE BALANCING
Best fitness
Evaluations (x 1000)
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50

NFL: No Free Lunch

All search algorithms are equivalent when compared over all possible discrete functions. Wolpert, Macready (1995)

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. Resampling is ignored. 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" versus a "WorstFirst" local search with restarts. For every j there exists an l such that

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

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).

POSSIBLE ALGORITHMS		POSSIBLE 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

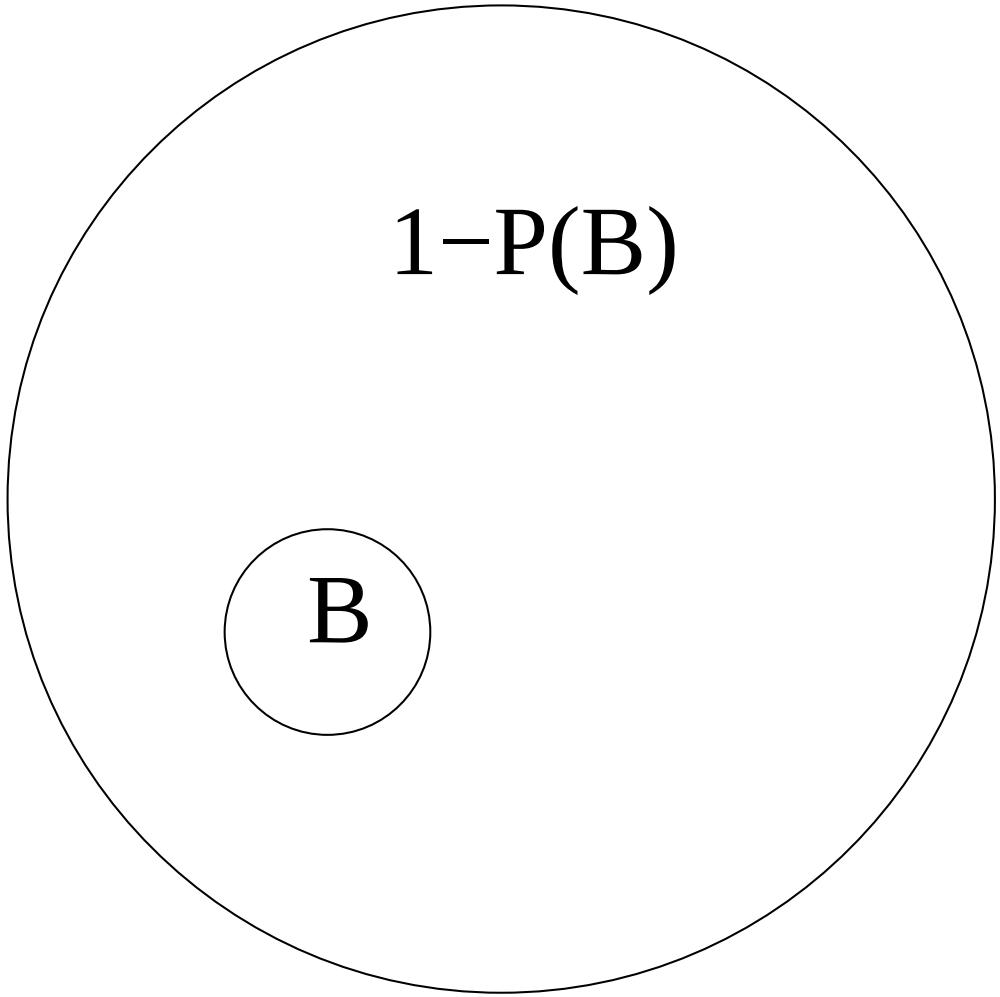
QUESTION:

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* $P(\beta) - \beta$.

This is True in the aggregate, but not on average.

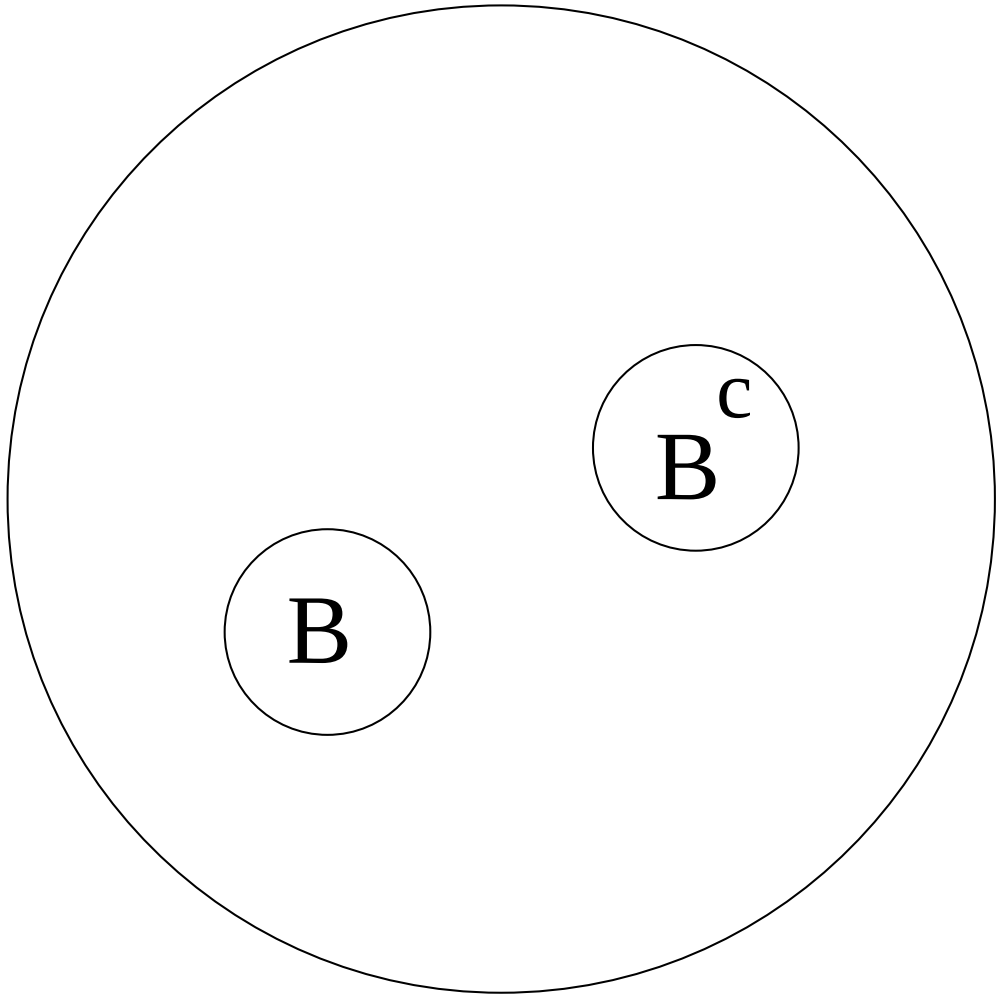

1−P(B)
B

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

The metafunction “On” allows us to construct a complementary benchmark suite β^c.

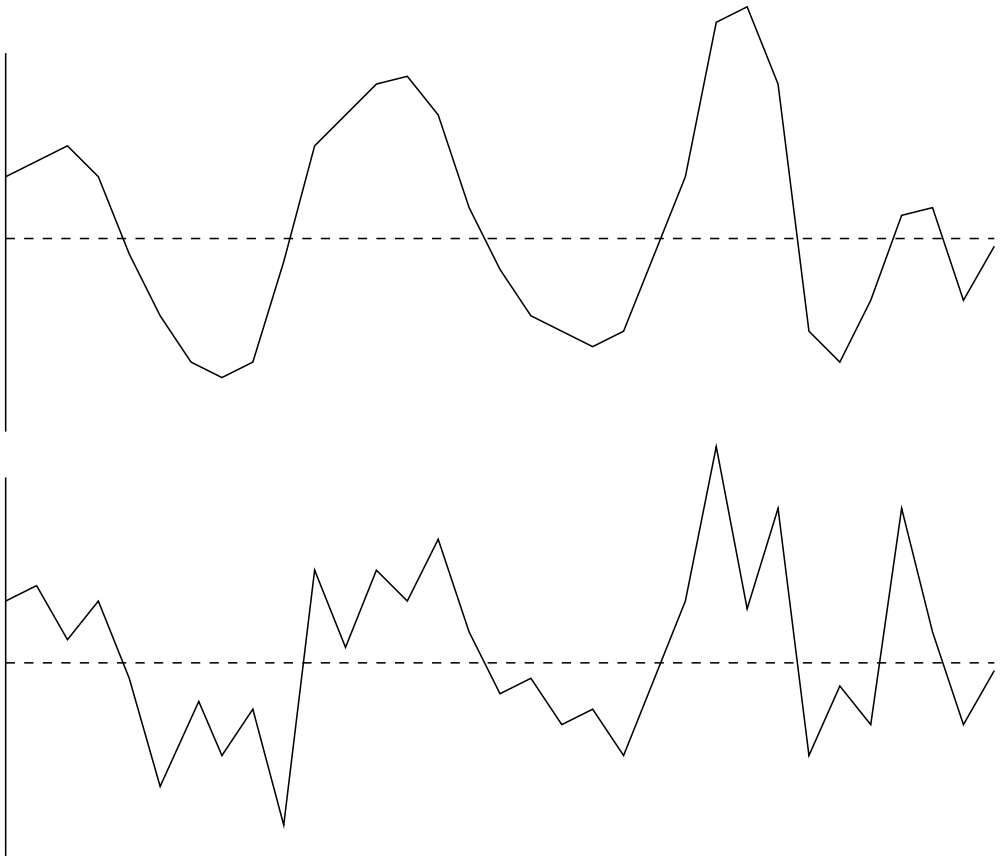
Given Algorithms A_i and A_k, for each problem $p \in \beta$ construct $p^c \in \beta^c$ such that

$$On(A_i, p) \equiv On(A_k, p^c)$$


B
B^c

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 (less than 0.5).

One way to beat random enumeration is to allocate a majority of sampled points to locations that are below threshold.

When does a simple bit climber using Binary or Gray Code representation beat random enumeration?

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 standard Binary code or 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

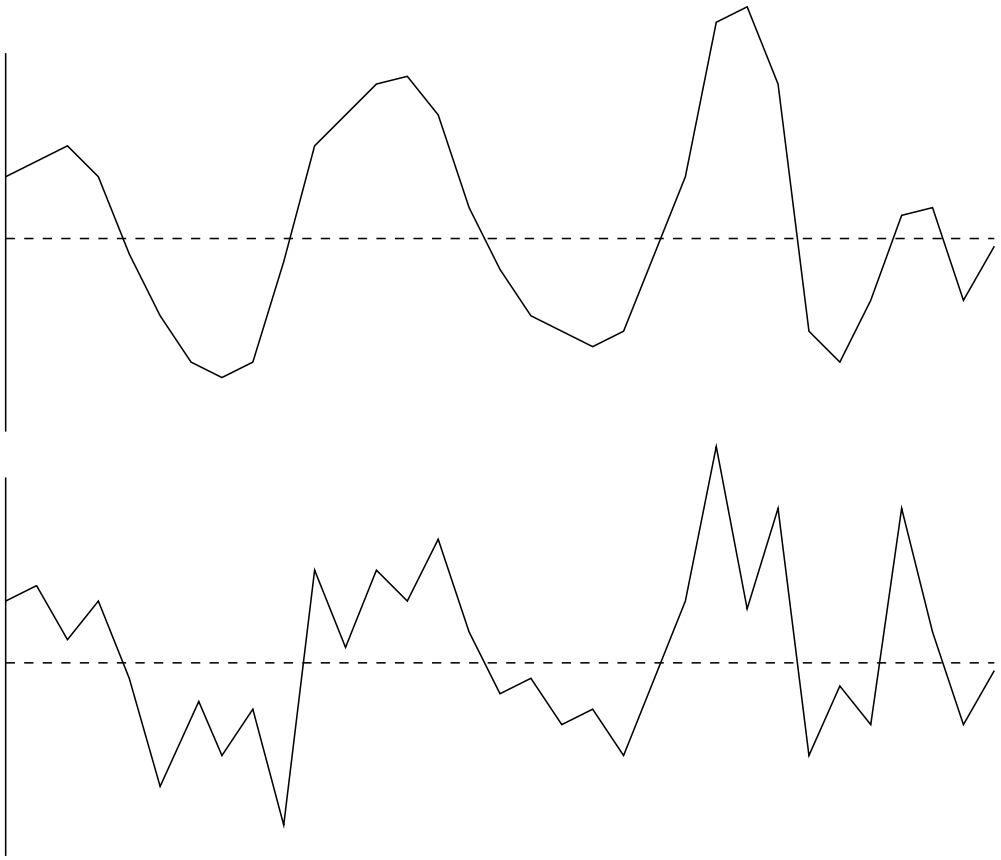
$$\lfloor (log(N/Q)) \rfloor - 1 > \log(Q) + 1$$

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 $\log(Q) + 1$.



Conclusions, Suggestions and Conjectures

- Black Box Optimization may be a lost cause.
- The best method is application specific.
- The next best method is problem class specific.
- Much of what we do is NOT Black Box Optimization.
- It's the neighborhood stupid!
  (with apologies to Bill Clinton and Dave Goldberg.)
- We need a wider view of Theory.
- We need more relevant and practical Theory.

Answer to the First Question:

I don’t know.