NSC : Non-Standard Computation

MEng 10 credit module

Susan Stepney,
John A. Clark, Sam Braunstein

Non-Standard Computation Group

what is *Standard* Computation?

- Turing paradigm
  - finite discrete classical state machine, Halting, Universal
  - closed system, predefined state space
- Von Neumann paradigm
  - sequential fetch-execute-store
- algorithmic paradigm
  - deterministic function from initial input to final output
  - black-box isolated from the world
- refinement paradigm
  - a known specification is refined to provably correct code
- pure logic paradigm
  - substrate (hardware/physics) is irrelevant

Non-standard views

- Real World as inspiration
  - natural computation : physics inspired, bio-inspired
- Real World as a computer
  - all computation and all data is embodied
    - physical effects – particularly quantum
  - analogue computation
    - the great missed opportunity of the 20^{th} Century?
- Open dynamic systems
  - no Halting, rather ongoing developing interactive processes
  - massive parallelism
    - "more is different"

“non-standard“ computation?

like defining the bulk of zoology
by calling it the study of
‘non-elephant animals’

– Stan Ulam (attrib)
(on the name “non-linear science”)

Biological complex adaptive systems

- evolution and genetics
  - competitive “survival of the fittest”
  - genetic algorithms, genetic programming
- immune systems
  - cooperative dynamics
- swarms, ants, termites
  - flocking, pheromones
- development and growth processes
  - L-systems, artificial embryology, ontogeny

$\Rightarrow$ Bio-inspired algorithms

embodiment of computation

- all computation, all data, is embodied
  - it must be realised in the Real World somehow
- therefore it obeys the laws of physics
  - mathematical models are abstractions from underlying physics
  - different physics $\Rightarrow$ different abstractions, different models
- the physical world is quantum mechanical
  - quantum weirdness : superposition, entanglement
- models of computation should encompass the quantum
  - then can exploit these weird properties
    - exponential speedup? teleportation?

$\Rightarrow$ Quantum Computing

“more is different”

- natural systems have vastly more than one atom, one molecule, one cell, one organism, one species, …
  - interacting in interesting ways
- systems with vastly more than one processing element
  - Ubiquitous (pervasive) computing
    - “chips with everything”
  - Agent systems
    - elements move, learn, adapt
  - Cellular Automata
    - emergent structures : from Gliders to UTM in Conway's Life
- FPGAs (Field Programmable Gate Arrays)

$\Rightarrow$ Massive parallelism, and emergence

open dynamical networks

- computation as a dynamic process
- far-from-equilibrium, heterogeneous, unstructured, metadynamic
  - continual learning and development – no "end point"
- phase space attractors, computational trajectories
  - autocatalytic chemical networks, cytokine immune network, genomic control networks, ecological webs, social and technological networks
- computation at the "edge of chaos"
- self organisation

$\Rightarrow$ Dynamical algorithms, and emergence

course overview : lectures

- 1. Introduction
- 2. Local Search
- 3-8. Bio-inspired population search and optimisation
  - evolutionary algorithms
  - swarms, ants
  - Artificial Immune Systems
  - growth and development
- 10-13. Embodied Computation
  - Quantum computation and communication
  - Computation by the real world : DNA, cells, membranes, chemicals, …
  - Analogue computation
- 14-18. Computational Dynamics, Complexity, and Emergence
  - fractals, Cellular Automata, self organisation
  - phase space, attractors, trajectories, network models

course overview : practicals

- discussion groups

1. (w3) assumptions of classical computation
2. (w6) reality as inspiration, not constraint
3. (w8) embodied issues
   - Quantum computing – niche or mainstream?
4. (w9) fractals and chaos
5. (w10) emergence and dynamics
   – plus course review, assessment handout

course overview : resources

- lecture notes
- applets
- links

- available on the course Web site

  http://www-course.cs.york.ac.uk/nsc/

NSC

Search and optimisation

Lecture overview

- solution and search spaces
  - objective function, fitness and cost functions
- fitness landscapes
  - local and global optima
  - ruggedness, hypercubes, NK-landscapes
  - data representations
- classification and clustering as search
- No Free Lunch theorem
  - what it means, and when it doesn't hold

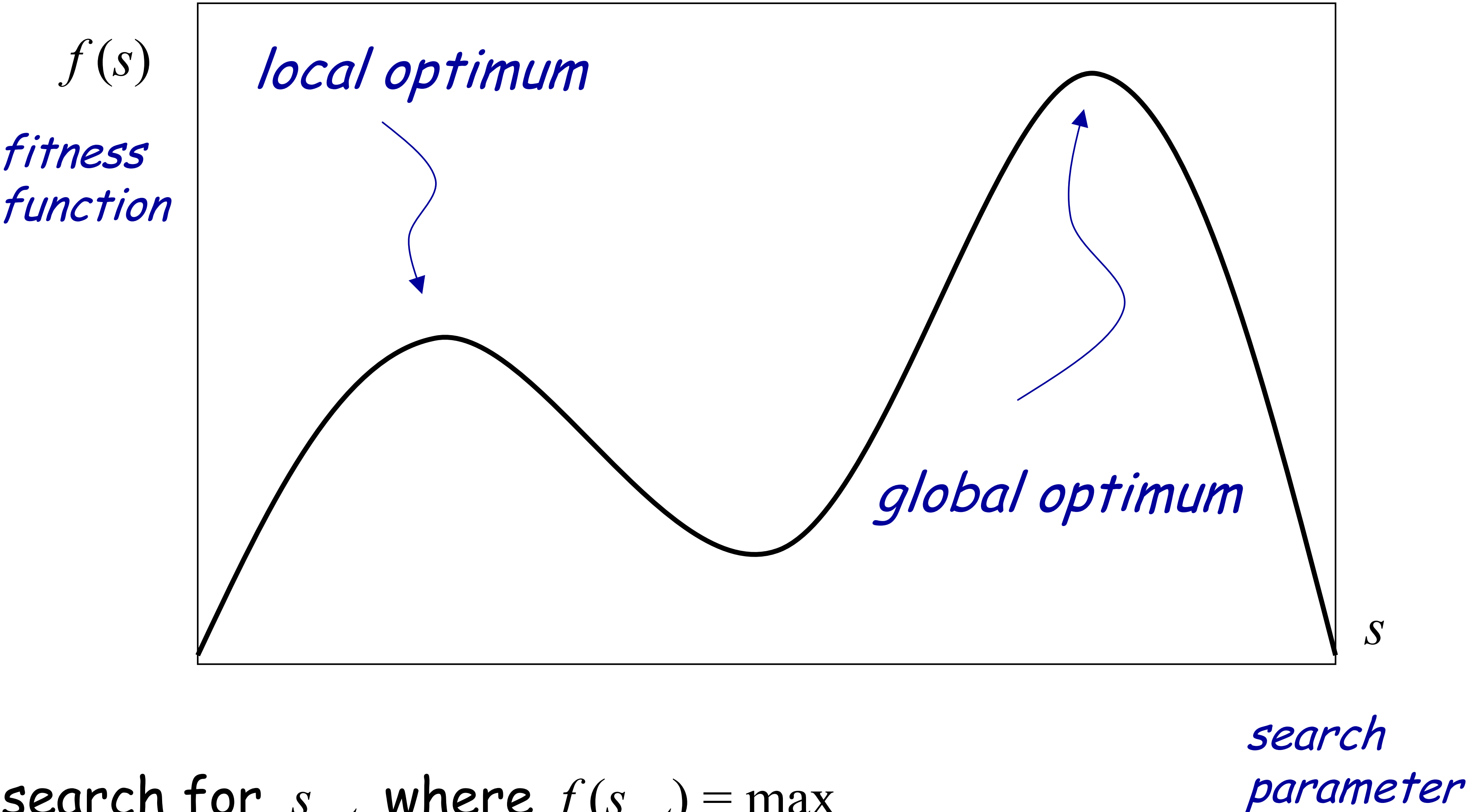
solution space

- solution space Σ
  - space of artefacts: programs, circuits, music, …
  - *objective function* defined on solution space, $\phi : \Sigma \to \Re$
    - multi-objective vector, $\varphi : (\Sigma_i \to \Re)^n$
  - the objective function measures the actual real world property to be optimised (maximised or minimised)
    - best power consumption
    - shortest path length
    - most melodious music
    - …
  - objective may be difficult to capture or quantify
    - what is the SI unit of melodious music?

search space

- search space S
  - encode the solution space in a form *suitable for search*
    - ***fitness (or cost) function*** defined on search space, $f : S \to \Re$
      - some implementations require the measure to be positive
    - *fitness* for maximum, *cost* for minimum (but not consistent)
  - optimising the fitness should also optimise the objective!
    - the choice of fitness function is a modelling decision
    - it can be scaled, inverted, smoothed, wrt to the objective
- algorithm to search that (very large) space
  - efficient algorithm will sample only a very small part of the search space, yet find good (high fitness, low cost) solutions
    - by exploiting structure of the search space
  - decode search result(s) back into solution space, $\Gamma : S \to \Sigma$

search landscape terminology

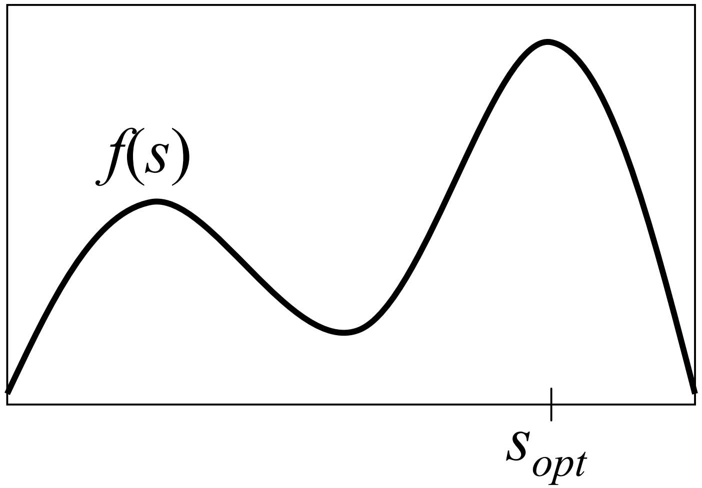


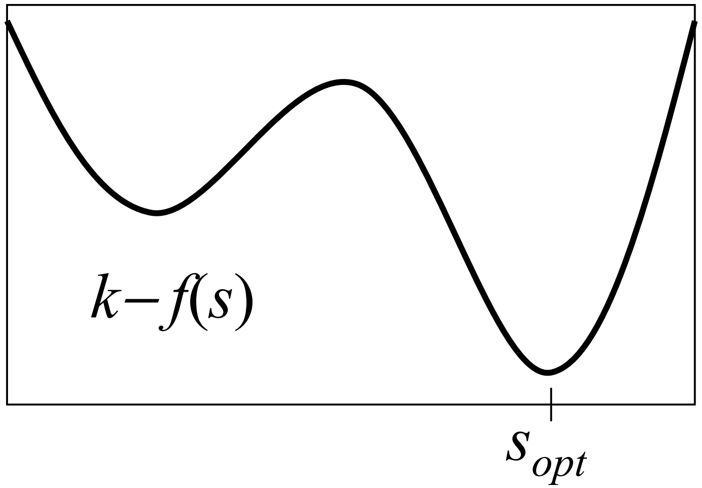
search for s_{opt} where $f(s_{opt}) = \max$

maximise or minimise?

- convert a maximisation problem to a minimisation problem by negating the fitness function
  - and adding an offset, to make the cost function positive, if necessary
    - choice of offset can affect behaviour of the search algorithm
      - as can adding a constant to fitness/cost function for any reason

maximise → *minimise*

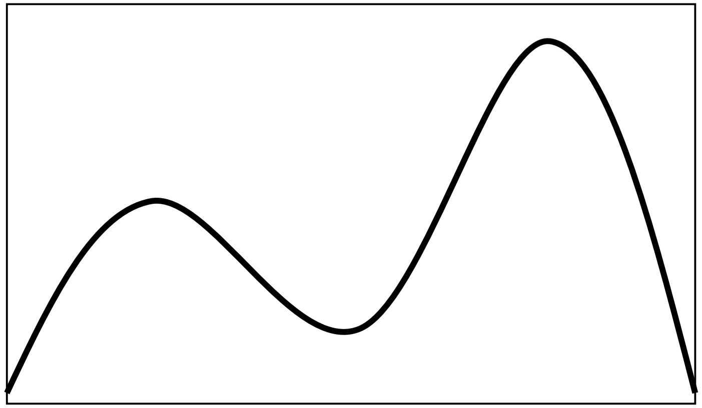




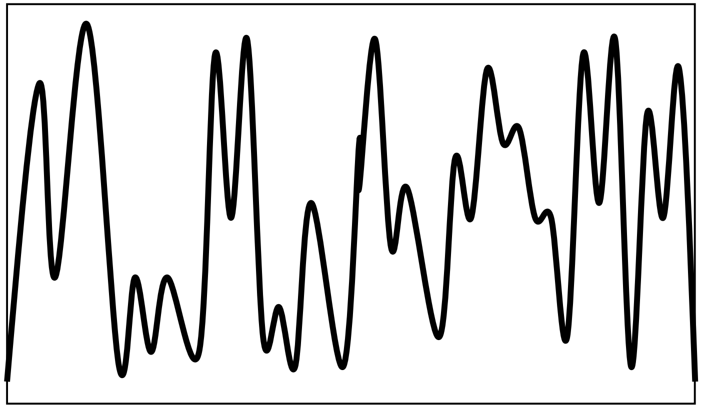
satisficing solutions

- don't necessarily need the *best* solution, just a "good enough" solution
- interested in ***satisficing***, rather than optimising
  - look for satisfactory solutions, that satisfice (minimally satisfy) the requirements, rather than the best, or optimal solution
  - if current solution is good enough, it doesn't matter that it may be very hard to get any better

search landscape examples

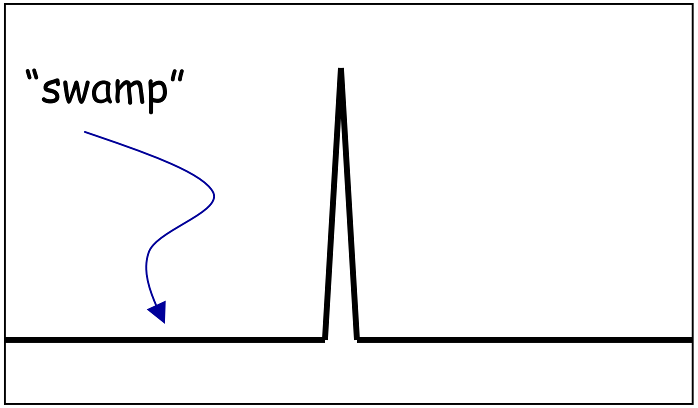


smooth landscape

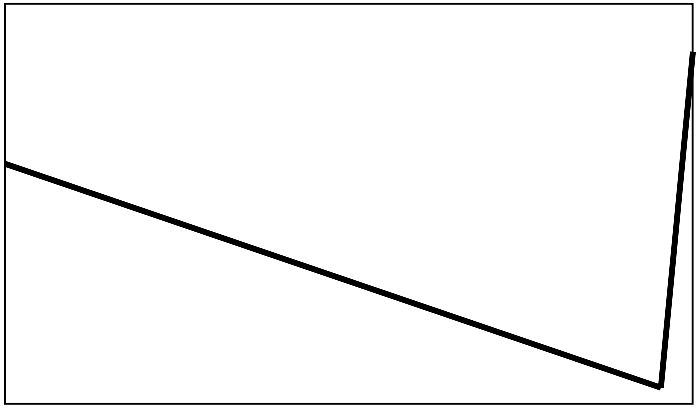


rugged landscape

small change in search parameter
-> large change in fitness

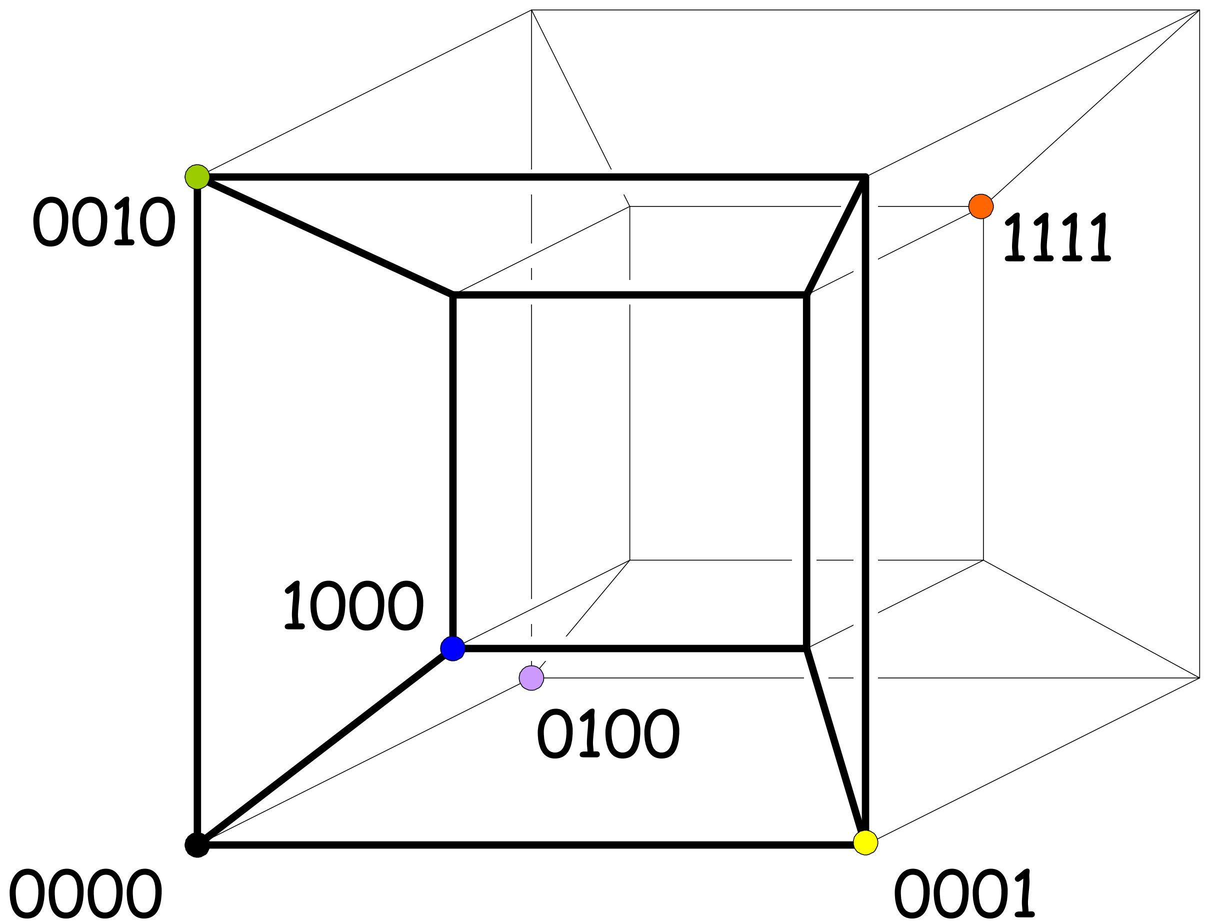


"needle in a haystack"



deceptive "trap" landscape

hypercube landscapes

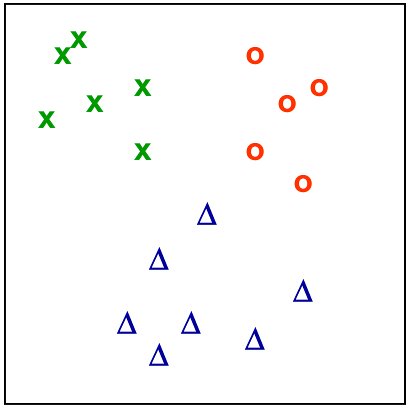


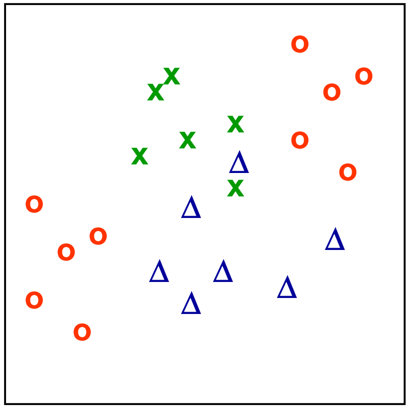
NK landscapes

- Kauffman's parameterised correlated fitness landscapes
- consider N dimensions of binary *traits* : N D hypercube
  - where overall fitness depends on correlations of traits
  - draw a network that connects each trait to all the other traits that affect its fitness
- NK model: N traits, fitness of each affected by K other "input" traits
  - hence the fitness of a trait depends on the values of $K+1$ traits
  - attempt to maximise total fitness (of all N traits)
  - conflicting constraints on maximising fitness of traits that depend differently on *same* input traits
    - conflicts increase as K increases
    - high $K \Rightarrow$ more rugged landscape
    - $K = 0$ = single peak ; $K = N - 1$ = fully random landscape

classification and clustering as search

- classification
  - group a population into "similar" sub-classes
    - clusters in parameter space, expressed as rules, or boundaries





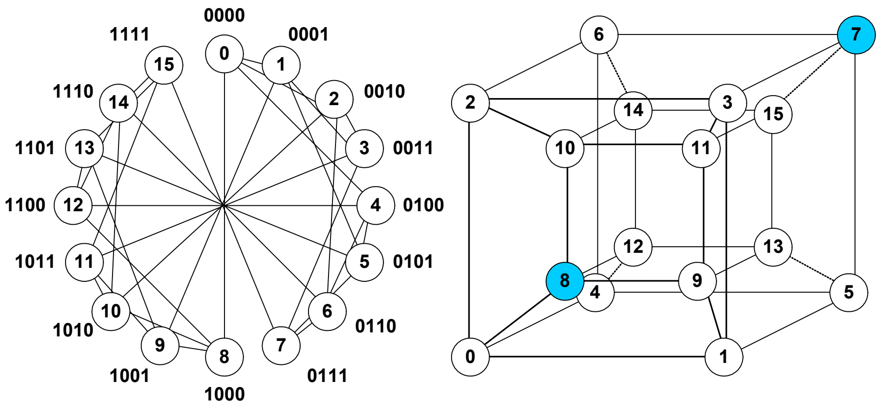
  - supervised : given predetermined sub-classes, algorithm finds boundaries
  - unsupervised : algorithm discovers sub-classes, too
- search, for a "fit" set of clustering rules

representation

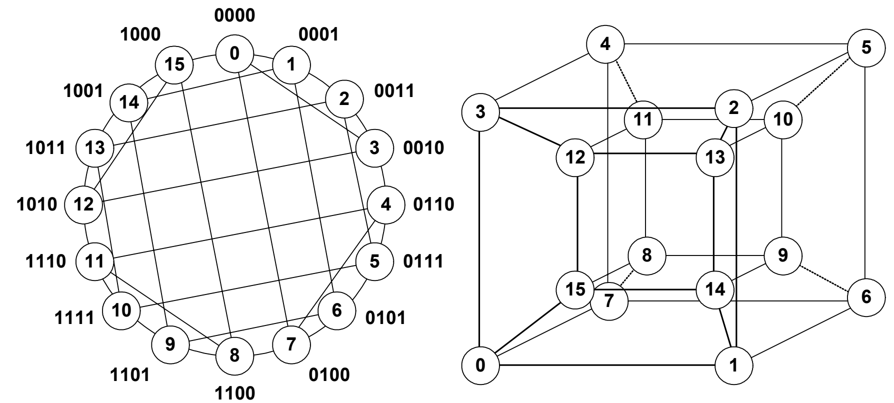
- choice of search space representation to fit the problem naturally, and be searchable
  - bit strings of length l : $S = \{0,1\}^l$
    - directly encode parameter values being optimised
  - more structured strings
    - integers, characters, structs, …
      - example: the component values in a fixed topology electronic circuit
  - finite state machines
    - to predict the next value in a sequence
  - computer programs
    - execute the program to generate (representation of) solution
      - example: draw a variable topology electronic circuit diagram
- a change of representation can "smooth" the search landscape, or make it more searchable in other ways

representation : Gray coding

- normal binary *v* Gray coding of integer bit strings
  - binary : flipping high bits has a bigger effect than low bits
  - Gray : consecutive underlying numbers differ by only one bit flip
  - Gray coding gives a much smoother search landscape
    - smoother, more continuous, adjacency relationship; fewer peaks
      - but may smooth out important features



binary adjacency



Gray adjacency

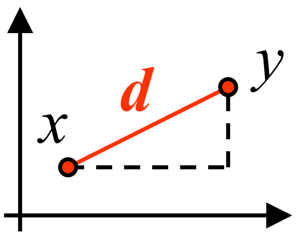
representation : data transformations

- change of basis so structure becomes clearer
  - "rotations" or scale changes; eigenvectors
- standard data transforms
  - Fourier / Laplace / …
- projecting onto a lower dimensional space (smaller)
  - might lose some information
  - change of representation might result in some "fixed" parameters that can be eliminated
- embedding in a higher dimensional space (smoothing)
  - discrete $\rightarrow$ continuous (real valued) $\rightarrow$ complex
- indirect encodings
  - as programs that generate results

distances

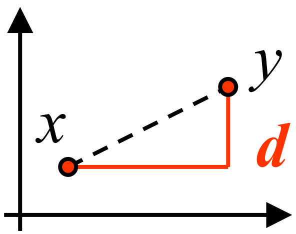
- distance between two points in an N-D space
- Euclidean distance
  - "straight line"
  - other "non-geometric" powers can also work

$$d = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$



- Manhattan distance
  - "city blocks"
  - cheap to calculate

$$d = \sum_{i=1}^{N} |x_i - y_i|$$



- Hamming distance
  - bitwise distance between strings

$$d = \sum_{i=1}^{N} (x_i \neq y_i)$$

"No Free Lunch" (NFL) results

- to do with the impossibility of finding a search algorithm effective over *all* landscapes
- definitions
  - search space S, fitness space R (where S, R are *finite* sets)
  - fitness function $f : S \rightarrow R$
  - search trace (or trajectory) $T_m = \langle (s_1, r_1), \ldots, (s_m, r_m) \rangle$
  - search algorithm $A : T_m \rightarrow S$ gives the "move" : next point to search
  - $T(A, f)$ = full search trace generated by A on f
  - performance measure $M : T \rightarrow \Re$
    - given a set of cost functions F, $M(A) \equiv \sum_{f \in F} M(T(A, f))$
- then, a NFL result applies to F, iff

$$\forall m : M ; a, b : A \bullet m(a) = m(b)$$

"No Free Lunch" theorems

a NFL result applies to the following Fs:

- F = (finite) set of all functions f from S to R
  - [Wolpert & Macready]
- F = (finite) set of all functions f "closed under permutation"
  - all functions that have the same set of results
    - $f_1 = \{(a,\alpha),(b,\alpha),(c,\delta)\}, f_2 = \{(a,\alpha),(b,\delta),(c,\alpha)\}, f_3 = \{(a,\delta),(b,\alpha),(c,\alpha)\}$
      - [Schumacher *et al.*]

D. H. Wolpert, W. G. Macready. No free lunch theorems for search. SFI-TR-95-02-010, Santa Fe Institute, 1995.

D. H. Wolpert, W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Comp.* **1**(1):67-82, 1997.

C. Schumacher, M. D. Vose, L. D. Whitley. The No Free Lunch and problem description length. *GECCO 2001*, 565-570, Morgan Kaufmann, 2001.

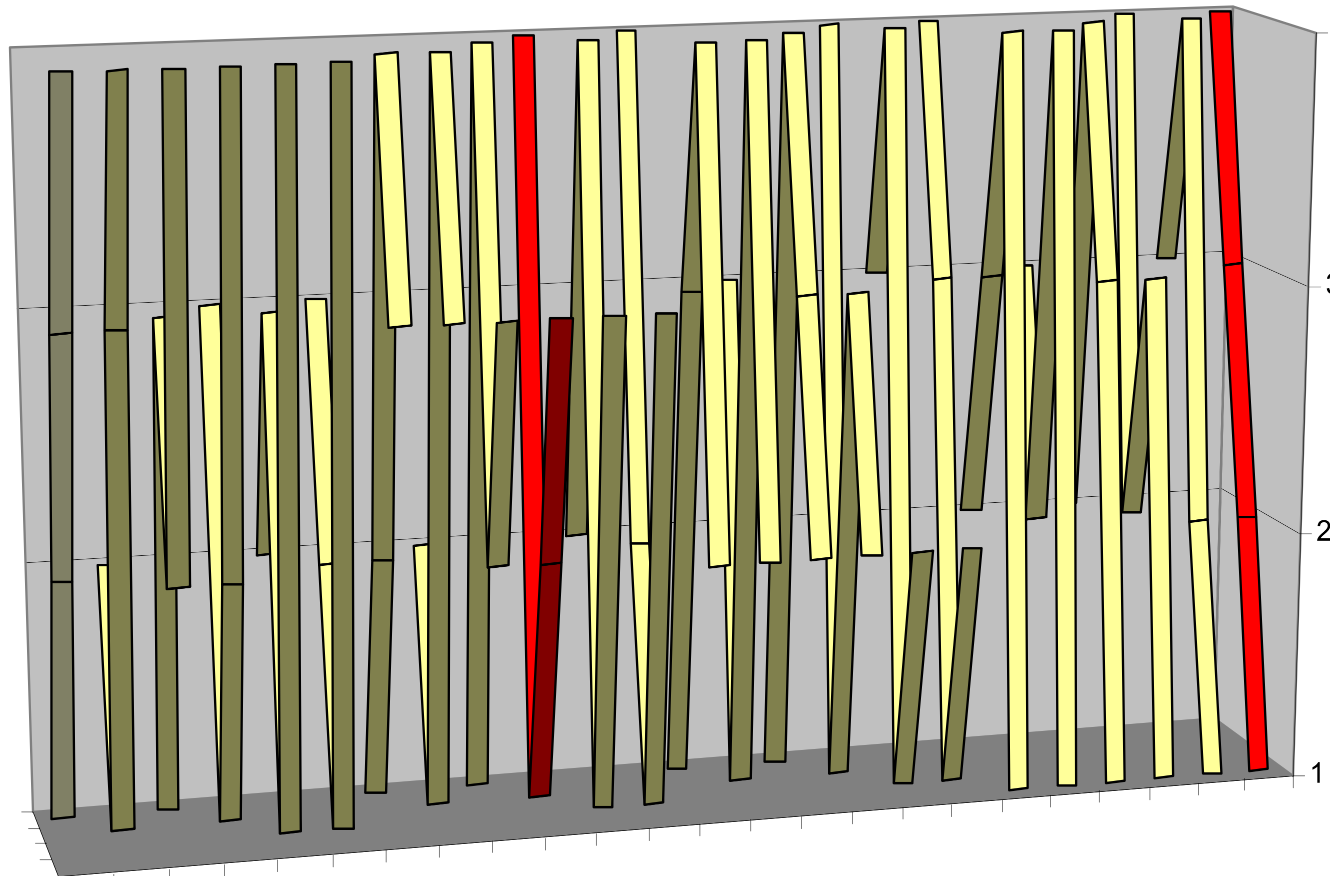
"No Free Lunch" in words

- any algorithm that searches for an optimum of a cost function performs *exactly the same* as any other, *when averaged over all cost functions*
  - random search is as good as anything else, on average

- so, if algorithm *A* is better than algorithm B on some cost functions, then there are other cost functions where B is better than *A*
  - in particular, B could be intuitively "wrong" (eg, using hill-*climbing* to find a *minimum*)
    - search algorithms look for global maxima based on information from other parts of the fitness landscape
    - for any given algorithm, there are many "deceptive" landscapes

"No Free Lunch" in pictures

1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2
2	1	3	4
2	1	4	3
2	3	1	4
2	3	4	1
2	4	1	3
2	4	3	1
3	1	2	4
3	1	4	2
3	2	1	4
3	2	4	1
3	4	1	2
3	4	2	1
4	1	2	3
4	1	3	2
4	2	1	3
4	2	3	1
4	3	1	2
4	3	2	1



all permutations

NFL: theoretically important

- NFL is a fundamental theoretical result
  - like undecidability or Halting
  - there is no general-purpose search algorithm any better than random search on average
    - NFL theorems hold for *exponentially large* sets of cost functions, most of which are "random" or uncomputable
    - NFL does not hold for sets of cost functions with bounded description lengths

M. J. Streeter. Two broad classes of functions for which a No Free Lunch result does not hold. *GECCO 2003*, LNCS 2724, 1418-1430, Springer, 2003.

- a "Gödel fallacy"
  - consider the Halting Problem v. proofs of program termination
    - interested in a particular class of all possible programs
    - can structure with loop variants, etc

NFL: irrelevant in practice

- in practice, we are not interested in *arbitrary* problems
  - we are interested in a *particular class* of search spaces
    - real world problems, not artificial "pathological" test functions
    - can always invent a test function that performs badly -- are these ones found in practice?
    - real world problems often have deep and interesting structure
- NFL demonstrates the importance of understanding the particular problem
  - can use *domain knowledge* to choose good search algorithms
    - "any algorithm performs only as well as the knowledge concerning the cost function put into the cost algorithm"

[Wolpert & Macready, 1995]