NSTC: Non-Standard Computation

3rd year 20 credit module

Susan Stepney

feat. Jon Timmis, Simon O’Keefe, Stefano Pirandola, Carlo Ottaviani

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 20th 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

⇒ 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)

⇒ 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

⇒ Dynamical algorithms, and emergence
module overview : lectures

L1. Introduction

L2-8. Search and optimisation: bio-inspired / physics inspired
   - local search
   - population based: evolutionary algorithms
   - population based: swarms, ants, Artificial Immune Systems (JT)
   - growth and development

L9-13. Embodied Computation
   - analogue computation
   - computation by real world: DNA, cells, membranes, chemicals, ... (SOK)
   - quantum computation (CO)
   - hypercomputation

L14-18. Computational Dynamics, Complexity, and Emergence
   - fractals, Cellular Automata, self organisation
   - phase space, attractors, trajectories, network models
module overview: group seminars

- **part a**: discuss a scenario (in a seminar session)
- **offline**: research ideas; prepare presentations (in groups)
- **part b**: present results (in a seminar session)

S1. assumptions of standard computation
S2. representations and cost functions
S3. analogue computing

-------

S4. quantum computation (SP)
S5. embodied computation
S6. assumptions of non-standard computation
module overview : practicals

P1. local search
P2. evolutionary algorithms
P3. L-systems
P4. quantum random walks
P5. Iterated Function Systems
P6. Cellular Automata
P7. reconstructing the attractor
module overview : assessment

• 2 part open assessment
  ▪ you will have to do your own literature research to answer these
  ▪ lectures / seminars / practicals provide only the starting point
• part A: 20% of overall mark
  ▪ 5pp report on a population-based algorithm
  ▪ handin Aut/10/Wed (4 Dec 2013)
  ▪ provides feedback on style/depth of answer expected for part B
• part B: 80% of overall mark
  ▪ 20pp report on two non-standard computers
  ▪ handin Sum/4/Wed (14 May 2014)
• both issued online at start of module (available now!)
module overview: resources

- lecture notes
- applets
  - lots of examples: play with these
- links
  - lots of examples: explore these

available on the module Web site

http://www-module.cs.york.ac.uk/nstc/
NSTC

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 : $\Sigma$

- space of real world artefacts
  - programs, circuits, music, ...
- **objective function** defined on solution space, $\phi : \Sigma \rightarrow \mathbb{R}$
  - multi-objective vector, $\varphi : (\Sigma_i \rightarrow \mathbb{R})^n$
- 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: $S$

- model the solution space in a form suitable for search
  - fitness (or cost) function defined on search space, $f : S \rightarrow \mathbb{R}$
    - 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, $\Gamma : S \rightarrow \Sigma$
search and solution spaces

real world
solution space : $\Sigma$

objective function
$\phi : \Sigma \rightarrow \mathbb{R}$

$\Gamma(s_0)$

decode
$\Gamma : S \rightarrow \Sigma$

computer model
search space : $S$

fitness / cost function
$f : S \rightarrow \mathbb{R}$

$S_0$

| 0110 A | 42.1 red ...
| 1110 D | 13.8 green ...
| 1010 C | 91.4 blue ...
| ...   |
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

\[ f(s) \quad \rightarrow \quad k - f(s) \]
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

deceptive “trap” landscape

“swamp”

“needle in a haystack”
hypercube landscapes
beware intuition: 2D landscapes

nD: even harder!
non-intuitive dimensionality

• all the picture examples so far have been 1,2,3 D
  ▪ we have good intuitions in small dimensions
  ▪ we have very poor intuitions in high dimensions
    ❖ but that’s where the interesting problems live

• the behaviour can be very different
  ▪ eg: some algorithms work by covering the \( N \) D search space with small \( N \) D hyperspheres, and arguing about the degree of coverage
    ❖ the volume and surface area of hyperspheres in high dimensions is very non-intuitive
    ❖ volume of \( n \) D cube = \( r^n \)
    ❖ volume of \( n \) D sphere = \( \frac{\pi^{n/2} r^n}{\Gamma\left(\frac{n}{2} + 1\right)} \to 0 \) as \( n \to \infty \)
    ❖ also, number of “uphill” directions can be non-intuitive
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, ...
      - eg: 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
      - eg: 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 vs 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
distance between two points in an $N$ D space

- Euclidean distance
  - “straight line distance”
  - other “non-geometric” powers can also work
- Manhattan distance
  - “city blocks”
  - cheap to calculate
- Chebyshev distance
  - cheap to calculate
- Hamming distance
  - bitwise distance between strings
  - ... lots more!

Euclidean distance:

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

Manhattan distance:

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

Chebyshev distance:

$$d = \max |x_i - y_i|$$

Hamming distance:

$$d = \sum_{i=1}^{N} (x_i \neq y_i)$$
“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 $M : T \rightarrow \mathbb{R}$
    - 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

$$\forall m : M; a, b : A \cdot m(a) = m(b)$$
“No Free Lunch” theorems

a NFL result applies to the following $F$s:

- $F =$ (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.]


“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

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

- 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: not a worry 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?
      • poor benchmarks are at best *meaningless*, at worst *misleading*
    ◆ “deceptive” functions are *brittle* to a change in representation

• 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]
  – real world problems often have deep and interesting structure