11 Novel Methods

- Evolutionary strategies
- Coevolutionary algorithms
- Ant algorithms
- The “No Free Lunch” theorem

11.1 Evolutionary Strategies

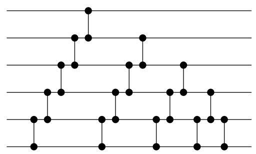
- Evolutionary methods for continuous optimisation (Bienert, Rechenberg, Schwefel et al. 1960’s onwards). Unlike GA’s, some serious convergence theory exists.
- Goal: maximise objective function $f : \mathbb{R}^n \to \mathbb{R}$. Use population consisting of individual points in $\mathbb{R}^n$.
- Genetic operations:
  - Mutation: Gaussian perturbation of point
  - Recombination: Weighted interpolation of parent points
  - Selection: Fitness computation based on f. Selection either completely deterministic or probabilistic as in GA’s
- Typology of deterministic selection ES’s (Schwefel):
  - Population size μ. λ offspring candidates generated by recombinations of μ parents.
  - $(\mu+\lambda)$-selection: best μ individuals from μ parents and λ offspring candidates together are selected.
  - (μ,λ)-selection: best μ individuals from λ offspring candidates alone are selected; all parents are discarded.

11.2 Coevolutionary Genetic Algorithms (CGA)

- Hillis (1990), Paredis et al. (from mid-1990’s)
- Idea: coevolution of interacting populations of solutions and tests/constraints as “hosts and parasites” or “prey and predator”
- Goals:
  1. Evolving solutions to satisfy a large & possibly implicit set of constraints
  2. Helping solutions escape from local minima by adapting the “fitness landscape”

Coevolution of sorting networks (1/3)

- Sorting networks: explicit designs for sorting a fixed number n of elements
- E.g. sorting network representing “bubble sort” of $n = 6$ elements:



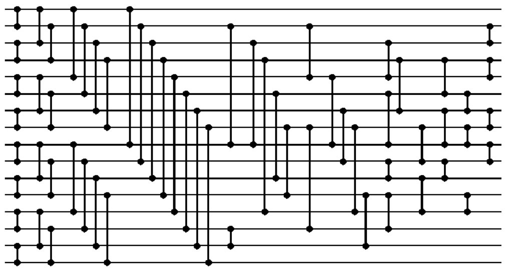
- Interpretation: elements flow from left to right along lines; each connection (“gate”) indicates comparison of corresponding elements, so that smaller element continues along upper line and bigger element along lower line
- Quality measures: size = number of gates (comparisons), depth (“parallel time”)

Coevolution of sorting networks (2/3)

- Quite a bit of work in the 1960's (cf. Knuth Vol. 3); size-optimal networks known for $n \leq 8$; for $n > 8$ the optimal design problem gets difficult.
- "Classical" challenge: $n = 16$. A general construction of Batcher & Knuth (1964) yields 63 gates; this was unexpectedly beaten by Shapiro (1969) with 62 gates, and later by Green (1969) with 60 gates. (Best known network.)
- Hillis (1990): Genetic and coevolutionary genetic algorithms for the $n = 16$ sorting network design problem:
  - Each individual represents a network with between 60 and 120 gates
  - Genetic operations defined appropriately
  - Individuals not guaranteed to represent proper sorting networks; behaviour tested on a population of test cases
  - Population sizes up to 65536 individuals, runs 5000 generations

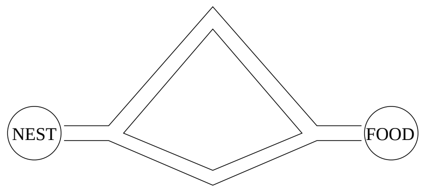
Coevolution of sorting networks (3/3)

- Result when population of test cases not evolved: 65-gate sorting network
- Coevolution:
  - Fitness of networks = % of test cases sorted correctly
  - Fitness of test cases = % of networks fooled
  - Also population of test cases evolves using appropriate genetic operations
- Result of coevolution: a novel sorting network with 61 gates:



11.3 Ant Algorithms

- Dorigo et al. (1991 onwards), Hoos & Stützle (1997), ...
- Inspired by experiment of real ants selecting the shorter of two paths (Goss et al. 1989):



- Method: each ant leaves a *pheromone trail* along its path; ants make probabilistic choice of path biased by the amount of pheromone on the ground; ants travel faster along the shorter path, hence it gets a differential advantage on the amount of pheromone deposited.

Ant Colony Optimisation (ACO)

- Formulate given optimisation task as a path finding problem from source s to some set of valid destinations $t_1, \ldots, t_n$ (cf. the A^* algorithm).
- Have agents ("ants") search (in serial or parallel) for candidate paths, where local choices among edges leading from node i to neighbours $j \in N_i$ are made probabilistically according to the local "pheromone distribution" τ_{ij}:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{j \in N_i} \tau_{ij}}.$$

- After an agent has found a complete path π from s to one of the t_k, "reward" it by an amount of pheromone proportional to the quality of the path, $\triangle\tau \propto q(\pi)$.

- Have each agent distribute its pheromone reward $\triangle\tau$ among edges (i,j) on its path π: either as $\tau_{ij} \leftarrow \tau_{ij} + \triangle\tau$ or as $\tau_{ij} \leftarrow \tau_{ij} + \triangle\tau/\mathrm{len}(\pi)$.
- Between two iterations of the algorithm, have the pheromone levels "evaporate" at a constant rate $(1-\rho)$:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}.$$

ACO motivation

- Local choices leading to several good global results get reinforced by pheromone accumulation.
- Evaporation of pheromone maintains diversity of search. (I.e. hopefully prevents it getting stuck at bad local minima.)
- Good aspects of the method: can be distributed; adapts automatically to online changes in the quality function $q(\pi)$.
- Good results claimed for Travelling Salesman Problem, Quadratic Assignment, Vehicle Routing, Adaptive Network Routing etc.

- Several modifications proposed in the literature:
  (i) to exploit best solutions, allow only best agent of each iteration to distribute pheromone;
  (ii) to maintain diversity, set lower and upper limits on the edge pheromone levels;
  (iii) to speed up discovery of good paths, run some local optimisation algorithm on the paths found by the agents;
  etc.

An ACO algorithm for the TSP (1/2)

- Dorigo et al. (1991)
- At the start of each iteration, m ants are positioned at random start cities.
- Each ant constructs probabilistically a Hamiltonian tour π on the graph, biased by the existing pheromone levels. (NB. the ants need to remember and exclude the cities they have visited during the search.)
- In most variations of the algorithm, the tours π are still locally optimised using e.g. the Lin-Kernighan 3-opt procedure.
- The pheromone award for a tour π of length $d(\pi)$ is $\triangle\tau = 1/d(\pi)$, and this is added to each edge of the tour: $\tau_{ij} \leftarrow \tau_{ij} + 1/d(\pi)$.

An ACO algorithm for the TSP (2/2)

- The local choice of moving from city i to city j is biased according to weights:

$$a_{ij} = \frac{\tau_{ij}^{\alpha}(1/d_{ij})^{\beta}}{\sum_{j\in N_i} \tau_{ij}^{\alpha}(1/d_{ij})^{\beta},}$$

  where $\alpha, \beta \geq 0$ are parameters controlling the balance between the current strength of the pheromone trail τ_{ij} vs. the actual intercity distance d_{ij}.
- Thus, the local choice distribution at city i is:

$$p_{ij} = \frac{a_{ij}}{\sum_{j\in N_i'} a_{ij}},$$

  where N_i' is the set of permissible neighbours of i after cities visited earlier in the tour have been excluded.

11.4 The “No Free Lunch” Theorem

- Wolpert & Macready 1997
- Basic content: All optimisation methods are equally good, when averaged over uniform distribution of objective functions.
- Alternative view: Any nontrivial optimisation method *must* be based on assumptions about the space of relevant objective functions. [However this is very difficult to make explicit and hardly any results in this direction exist.]
- Corollary: one cannot say, unqualified, that ACO methods are “better” than GA’s, or that Simulated Annealing is “better” than simple Iterated Local Search. [Moreover as of now there are *no* results characterising some nontrivial class of functions $\mathcal{F}$ on which some interesting method $\mathcal{A}$ would have an advantage over, say, random sampling of the search space.]

The NFL theorem: definitions (1/3)

- Consider family $\mathcal{F}$ of all possible objective functions mapping finite search space $\mathcal{X}$ to finite value space $\mathcal{Y}$.
- A *sample* d from the search space is an ordered sequence of distinct points from $\mathcal{X}$, together with some associated cost values from $\mathcal{Y}$:

$$d = \{(d^x(1), d^y(1)), \ldots, (d^x(m), d^y(m))\}.$$

  Here m is the *size* of the sample. A sample of size m is also denoted by d_m, and its projections to just the x- and y-values by d_m^x and d_m^y, respectively.
- The set of all samples of size m is thus $\mathcal{D}_m = (\mathcal{X} \times \mathcal{Y})^m$, and the set of all samples of arbitrary size is $\mathcal{D} = \cup_m \mathcal{D}_m$.

The NFL theorem: definitions (2/3)

- An *algorithm* is any function a mapping samples to *new* points in the search space. Thus:

$$a : \mathcal{D} \to \mathcal{X}, \quad a(d) \notin d^x.$$

- *Note 1:* The assumption $a(d) \notin d^x$ is made to simplify the performance comparison of algorithms; i.e. one only takes into account *distinct* function evaluations. Not all algorithms naturally adhere to this constraint (e.g. SA, ILS), but without it analysis is difficult.
- *Note 2:* The algorithm may in general be stochastic, i.e. a given sample $d \in D$ may determine only a *distribution* over the points $x \in \mathcal{X} - d^x$.

The NFL theorem: definitions (3/3)

- A *performance measure* is any mapping Φ from cost value sequences to real numbers (e.g. minimum, maximum, average). Thus:

$$\Phi : \mathcal{Y}^* \to \mathbb{R},$$

where $\mathcal{Y}^* = \cup_m \mathcal{Y}^m$:
- Finally, denote by $P(d_m^y \mid f, m, a)$ the probability distribution of value samples of size m obtained by using a (generally stochastic) algorithm a to sample a (typically unknown) function $f \in \mathcal{F}$.

- More precisely, such a sample is obtained by starting from some a-dependent search point $d^x(1)$, querying f for the value $d^y(1) = f(d^x(1))$, using a to determine search point $d^x(2)$ based on $(d^x(1), d^y(1))$, etc., up to search point $d^x(m)$ and the associated value $d^y(m) = f(d^x(m))$. The value sample d_m^y is then obtained by projecting the full sample d_m to just the y-coordinates.

The NFL theorem: statement

Theorem

[NFL] For any value sequence d_m^y and any two algorithms a_1 and a_2:

$$\sum_{f \in \mathcal{F}} P(d_m^y \mid f, m, a_1) = \sum_{f \in \mathcal{F}} P(d_m^y \mid f, m, a_2).$$

The NFL theorem: corollaries

Corollary

[1] Assume the uniform distribution of functions over $\mathcal{F}$, $P(f) = 1/|\mathcal{F}| = |\mathcal{Y}|^{-|\mathcal{X}|}$. Then for any value sequence $d_m^y \in \mathcal{Y}^m$ and any two algorithms a_1 and a_2:

$$P(d_m^y \mid m, a_1) = P(d_m^y \mid m, a_2).$$

Corollary

[2] Assume the uniform distribution of functions over $\mathcal{F}$. Then the expected value of any performance measure Φ over value samples of size m,

$$E(\Phi(d_m^y) \mid m, a) = \sum_{d_m^y \in \mathcal{Y}^m} \Phi(d_m^y) P(d_m^y \mid m, a),$$

is independent of the algorithm a used.