Natural Computing

Lecture 13

Michael Herrmann
mherrman@inf.ed.ac.uk
phone: 0131 6 517177
Informatics Forum 1.42

1/11/2011

Particle Swarm Optimisation and Metaheuristic Optimisation

The canonical PSO algorithm

For each particle (for all members in the swarm) $i = 1 \ldots n$

- Create random vectors $r_1,\ r_2$ with components in $U[0,1]$
- update velocities

$$v_i \leftarrow \omega v_i + \alpha_1 r_1 \circ (\hat{x}_i - x_i) + \alpha_2 r_2 \circ (\hat{g} - x_i)$$

$\circ$: componentwise multiplication

- update positions

$$x_i \leftarrow x_i + v_i$$

- update local bests (for a minimisation problem)
  $\hat{x}_i \leftarrow x_i$ if $f(x_i) < f(\hat{x}_i)$
- update global best (for a minimisation problem)
  $\hat{g} \leftarrow x_i$ if $f(x_i) < f(\hat{g})$

Relation to probabilistic methods

- Probabilistic methods are ofter based on assumptions (Gaussianity, optimal sampling, large sample size etc.) which often do not hold in practical applications
- Meta-heuristic approaches do well in many examples
  - Toy examples are often designed ad-hoc for a particular method and are thus unsuitable for a fair comparison.
  - Success in real-world examples depends much on domain knowledge, quality of analysis, iterative re-design etc.
- Meta-heuristic algorithms can include strict algorithms for local search
- Meta-heuristic algorithms can be used to initialize, adapt, optimize or tune the "exact" algorithms

Particle filters

- Given observations Y_k; reconstruct true states X_k
  Initial distribution: $p(X_0|Y_0) = p(X_0)$
  Markovian evolution:
  $p(X_k|Y_{1,\ldots k-1}) = \int p(X_k|X_{k-1})\, p(X_{k-1}|Y_{1,\ldots k-1})$
  Bayes' rule:
  $$p(X_k|Y_{1,\ldots k-1}) = \frac{p(Y_k|X_k)\, p(X_k|Y_{1,\ldots k-1})}{p(Y_k|Y_{1,\ldots,k-1})}$$
- Represent posterior distribution by N weighted samples (here: $w_i^k = \frac{1}{N}$) obtained from:
$$p(X_k|Y_{1,\ldots k-1}) \approx \frac{1}{N}\sum_{i=1}^{N} p_i(X_k):\;\; p(X_k|Y_{1,\ldots k}) \propto \sum_{i=1}^{N} p(Y_k|X_k)\, p_i(X_k)$$
- Problems: impoverishment and sample size effects (e.g. if the likelihood is concentrated at the tail of the prior)

G. Tong, Z. Fang, X. Xu (2006) A PS optimized PF for non-linear system state estimation. Proc. Congress on Evolutionary Computation, 438-442.

PSO PF

- Use PSO for sampling
- Standard PSO with Gaussian randomness in the velocity update ("Gaussian swarm")
- Fitness: $f \exp\left(-\frac{1}{2R_k}\left(y_{\text{new}} - y_{\text{pred}}\right)^2\right)$
- R_k: observation covariance
- Modulate weights: $w_i^k = w_i^{k-1} p\left(y^k | x_i^k\right)$, $w_i^k \leftarrow \frac{w_i^k}{\sum_{j=1}^N w_j^k}$
- Now represent posterior by weighted samples
- Avoids divergence and does well with small number of particles

G. Tong, Z. Fang, X. Xu (2006) A PS optimized PF for non-linear system state estimation. Proc. Congress on Evolutionary Computation, 438-442.

Comparison of GA and PSO

- Similar in some respects:
  - 1. Random generation of an initial population
  - 2. Calculation of a fitness value for each individual.
  - 3. Reproduction of the population based on fitness values.
  - 4. If requirements are met, then stop. Otherwise go back to 2.
- Modification of individuals
  - In GA: by genetic operators
  - In PSO: Particles modify themselves via their own velocity. Memory of their personal best. "Elitism" by global best.
- Sharing of information
  - Mutual in GA. Population moves in groups to optimal areas.
  - One-way in PSO: Source of information is gBest (or lBest). All particles tend to converge to the best solution quickly.
- Representation
  - GA: usually discrete
  - PS: usually continuous

www.swarmintelligence.org/tutorials.php

Discrete Particle Swarm Optimization

A particle in a swarm

- has a position and a velocity
- knows its position & objective function value for this position
- knows its neighbours, best previous position and objective function value (or: current position & objective function value)
- remember its best previous position

Behaviour determined by a compromise between three influences

- To follow its own way (momentum, “self-confidence”)
- To go towards its best previous position (“experience”)
- To go towards the best neighbour’s best previous position, or towards the best neighbour (“peer pressure”)

(see Maurice Clerc (Maurice.Clerc@WriteMe.com) http://www.mauriceclerc.net)

Canonical PSO

$x_i,\ v_i \in \mathbb{R}^d,\ 1 \leq i \leq n,\ r_1,\ r_2 \in \mathbb{R}^d\ ,\omega,\ \alpha_1,\ \alpha_2 \in \mathbb{R}^+,$

$f\ :\ \mathbb{R}^d \to \mathbb{R}^+$ to be minimized

For all member of the swarm

$$v_i := \omega v_i + \alpha_1 r_1 \circ (\hat{x}_i - x_i) + \alpha_2 r_2 \circ (\hat{g} - x_i)$$

$$x_i := x_i + v_i$$

$$\hat{x}_i := x_i \text{ if } f(x_i) < f(\hat{x}_i)$$

$$\hat{g} := x_i \text{ if } f(x_i) < f(\hat{g})$$

until termination criterion is met

Discrete PSO

States are implied by the optimisation problem, e.g. discrete: $s \in \mathbb{Z}$

- Option 1: Run the algorithm for continuous states x and discretize $[s = (\text{int})\left(x + \frac{1}{2}\right)]$ after a solution has been found
- Option 2: If the objective function does not accept continuous values then discretize before fitness evaluation
- Option 3: Use discrete states $s = x$. The velocities are still continuous but are incremented by discrete steps. When updating s with a small velocity there is no effect, only from a certain threshold s is actually changed. This could be advisable if continuous values of the states have no meaning
- Option 4: Use discrete states $s = x$ and continuous velocities, but rather consider velocities as probabilities of changing a state (in particular for binary states).
- Option 5: Use a more systematic approach (cf. below)

Example: Sequence alignment

Time-warped sequences $q_m(t)$, $m = 1, \ldots, M$, $t \in [0, T]$

Ideally, if we had the correct warping functions $w_m(t)$ for each sequence then for all t

$$q_1(t + w_1(t)) = \cdots = q_M(t + w_M(t))$$

More generally, we cannot assume exact equality, so we minimise

$$f[w] = \sum_{i,j=1}^{M} \int (q_i(t + w_i(t)) - q_j(t + w_j(t)))^2 \, dt$$

by choosing appropriate $w_m(t)$. This is an infinite-dimensional problem.

Example: Sequence alignment

Choose a discretization $t = 1, \ldots, T$ (or use the natural discretization of the data)

$$f[w] = \sum_{i,j=1}^{M} \sum_{t=1}^{T} \left(q_i\left(t + w_i(t)\right) - q_j\left(t + w_j(t)\right)\right)^2$$

w is a $M \times T$ dimensional vector that can be used as state x in PSO.

However, having discretized t only discrete values of w are meaningful. Nevertheless, all of the above options are applicable.

Note of caution:

For all options, adaptive discretisation schemes might be useful.

Also the parameters ω, α_1, α_2 may have optimal values far from the standard values for the continuous case.

Systematic approach to discrete PSO: Operator formalism

The PSO update rules require four operations:

- Subtraction (position – position) operator:
  two positions x_1 and x_2: $x_2 - x_1 = v$ (velocity)
- Multiplication (scalar coefficient $\times$ velocity) operator:
  learning coefficient: α, ω, velocity v : $c \times v$ (velocity)
- Addition (velocity $\oplus$ velocity) operator:
  two velocities: v_1 and v_2: $v_1 \oplus v_2$ (velocity)
- Addition (position $+$ velocity) operator:
  position x and v velocity: $x + v = x_1$ (position)

Example: Discrete PSO for TSP

Search space of positions/states $S = \{s_i\}$

Cost/objective function f on S maps into a set of values: $S \to C = \{c_i\}$

Order on C, or, more generally, a semi-order: either $c_i < c_j$ or $c_i \geq c_j$ (if comparable)

If we want to use a physical neighbourhood, we also need a distance d in the search space.

We are looking for Hamilton cycles in a weighted graph $G = \{E_G, V_G\}$

Enumber E_G and search for sequences of $N+1$ nodes with first and last identical, otherwise different.

$f(s) = \sum_{i=1}^{N} w_{n_i, n_{i+1}}$ with $n_{N+1} \equiv n_0$

Discrete velocities

- What is a state? A vector containing N nodes
- What is a velocity?
  - Define it as a permutation. Can be represented as a composition of permutations of pairs of nodes (swaps)
  - Simplest case: the exchange of two nodes: $(\ldots, i, \ldots, j, \ldots) \rightarrow (\ldots, j, \ldots, i, \ldots)$, i.e. the cycle: (ij).
  - More generally: $\{(i_k, j_k)\}_{k=1,\ldots,|v|}$
- A negative velocity? Subtraction as inverted sequence of swaps
  - -v=$\left\{\left(i_{|v|-k+1}, j_{|v|-k+1}\right)\right\}_{k=1,\ldots,|v|}$
- Adding a velocity to a state means applying a permutation (v) to a set of objects (x)

Further operations

- Difference between states?
  - permuation that transforms x_1 into x_2
- Sum of velocities?
  - perform first the pair exchances of v_1 than those of v_2 (not commutative; may be contracted into fewer pairs)
- Multiplication by a scalar?
  - $\omega = 0 \Rightarrow \omega v = Id$
  - $\omega \in (0, 1]$: remove all pairs from v after the $\lceil c\,|v|\rceil$-th swap.
  - $\omega > 1$ concatenate (int)$\lfloor c \rfloor$-times and continue with $\lceil c\,|v|\rceil$ pairs from the beginning of v

Algorithm: Discrete PSO

$$\begin{aligned} v_{t+1} &= \omega v_t \oplus \alpha_1 (x^* - x_t) \oplus \alpha_2 (p - x_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned}$$

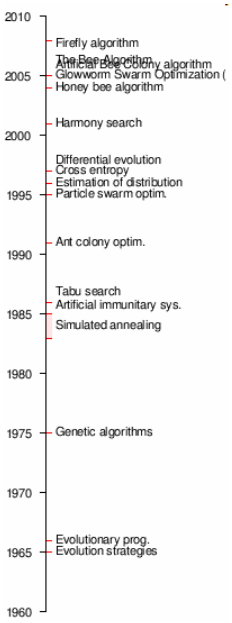
- e.g. $x^* - x_t$ is a velocity (transforming x_t into x^*)
- the three sets of swaps are concatenated (perhaps beginning with ωv_t, i.e. a part (for $\omega < 1$) of the swaps in v_t)
- the result is applied to the ordered set that is represented by x_t

How does this work?

- Often better than GA or ACO. Why?

Meta-Heuristic Search

- $\mu\varepsilon\tau\alpha$ “beyond”, $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ "to find“
- Applied mainly to combinatorial optimization
- The user has to modify the algorithm to a greater or lesser extend in order to adapt it to specific problem
- These algorithms seem to defy the no-free lunch (NFL) theorem due to the combination of
  - biased choice of problems
  - user-generated modifications
- Can often be outperformed by a problem-dependent heuristic



The General Scheme

1. Use populations of solutions/trials/individuals
2. Transfer information in the population from the best individuals to others by selection+crossover/attraction
3. Maintain diversity by adding noise/mutations/ intrinsic dynamics/amplifying differences
4. Avoid local minima (leapfrog/crossover/more noise/ subpopulations/border of instability/checking success, random insertions)
5. Whenever possible, use building blocks/partial solutions/royal road functions
6. Store good solutions in memory as best-sofar/ iteration best/individual best/elite/pheromones
7. Use domain knowledge and intuition for encoding, initialization, termination, choice of the algorithm
8. Tweak the parameters, develop your own variants

“Banal Metaheuristic”

*** in three easy steps ***

1. Call the user-provided state generator.

2. Print the resulting state.

3. Stop.

Given any two distinct metaheuristics M and N, and almost any goal function f, it is usually possible to write a set of auxiliary procedures that will make M find the optimum much more efficient than N, by many orders of magnitude; or viceversa. In fact, since the auxiliary procedures are usually unrestricted, one can submit the basic step of metaheuristic M as the generator or mutator for N.

en.wikipedia.org/wiki/Metaheuristic

Contra

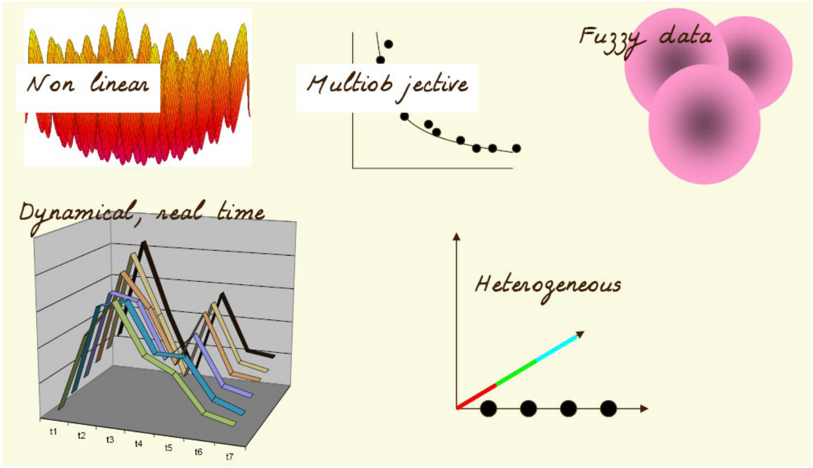
- No-free-lunch theorem implies that there must be some implicit assumptions that single out “good” problems (one such assumption is the correlation between goal function values at nearby candidate solutions)
- If these assumptions were made explicit more specific algorithms could be designed
- Random search often seems to be the essential component
- The quality of a ME algorithm is not well-defined because user-provided domain knowledge enters
- There are many “classical” problems which are fully understood and where ME algorithms perform comparatively poor. (LS is usually not state of the art)
- Dilettantism: A few hours of reading, thinking and programming can easily save months of computer time used up by ME

en.wikipedia.org/wiki/Metaheuristic

Pro

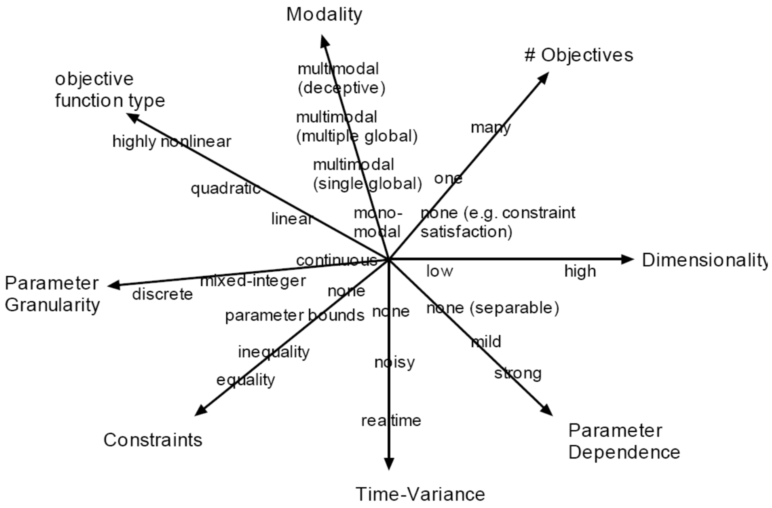
- If you know a better solution then why using ME? But if not, then why not?
- Its not just random search
- There are a number of applications where ME are performing reasonably well
- Theoretical expertise, problem analysis, modeling and implementation are cost factors in real-world problems
- There are domains where modeling is questionable, but the combination of existing solutions is possible (minority games, e.g. esthetic design, financial markets)
- Nature is an important source of inspiration
- It may help to understand decision making in nature and society

Ecological niches for MH algorithms



PSO Mini Tutorial on Particle Swarm Optimisation (2004), Maurice.Clerc@WriteMe.com

Some of the dimensions of the problems space



Metaheuristic algorithms: What do they do?

- Solving combinatorial optimization problems
- Iteratively improving candidate solutions
- Few assumptions about the problem
- Usually population of candidate solutions
- Some mechanism of accumulation of information about the problem

Definition of a Combinatorial Optimization Problem (COP)

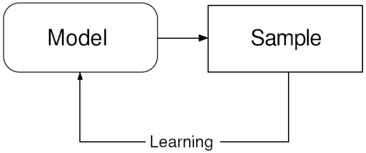
(S, Ω, f)

- S is a search space defined over a finite set of discrete decision variables
- S is contained in $D = \{v_1, \dots, v_D\}$
- Ω is a set of constraints among the variables
- S_Ω: set of solutions that satisfy all constraints
- f is an objective function to be maximised
- Optimum (global maximum):
  $s^* \in S_\Omega : f(s^*) \geq f(s) \ \forall s \in S_\Omega$
- Task: Find at least one optimum

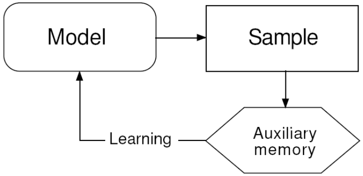
GA/GP and ACO as COPs

	GA/GP	ACO
S	bit strings/trees	paths in a graph
Ω	correctness (GP)	e.g. non-intersecting
f	fitness	total path length
S_Ω	correct programs	e.g. non-intersecting paths
opt.	fittest individual best program on fitness cases	path of minimal length

Model-Based Search
Framework for expressing relation between algorithms



Scheme of the MBS approach



MBS approach with memory

E.g. in ACO:

- Model: pheromone matrix
- Sample: ants following pheromone traces
- Learning: pheromone update
- Auxilary memory: best-so-far solution

Zlochrin, Birattari, Meuleau, Dorigo: Model-based Search for Combinatorial Optimization: A Critical Survey. Annals of Operations Research 2004.

Model Based Search

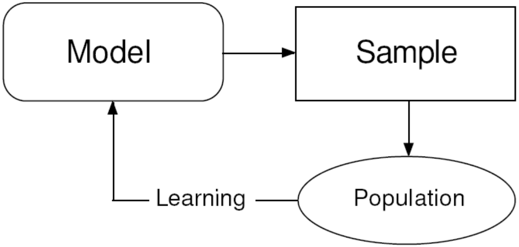
- Instance-based, i.e. improvement based on previous instance(s) or
- model-based: Candidate solutions are constructed using some parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is deemed to bias future sampling toward low cost solutions.
- Models enable theoretical predictions.

ACO as MBO

- A finite set $C = c_1, c_2, \ldots, c_n$ of components (n is the number of components)
- A finite set X of states of the problem, where a state is a sequence $x = c_i, c_j, \ldots, c_k, \ldots$ over the elements of C. The length of sequence x, i.e., the number of components in the sequence, is expressed by $|x|$. The set of (candidate) solutions S is a subset of X (i.e. $S \subseteq X$).
- A set of feasible states X_f, with $X_f \subseteq X$, defined via a set of constraints Ω
- A non-empty set S^* of optimal solutions, with $S^* \subseteq X_f$ and $S^* \subseteq S$
- Formulation of the update in the hyper-cube framework
- Result is a fully-connected weighted graph

GA as MBS

- GA seems to be instance-based, but samples are not drawn independently. Dependencies can be captured by a model:
- Generate new solutions using the current probabilistic model
- Replace (some of) the old solutions by the new ones.
- Modify the model using the new population.



GA as MBS

compact Genetic Algorithm (cGA) (Harik et al., 1999)

- Probabilistic (so far not different from instance-based) simulation of a GA with tournament selection:
- Probabilistic model of the population: individuals are generated by biased draws based on a probability vector. E.g. if the vector entry p_i is 0.9 it is likely to have a 1 at position i in this individual's string.
- Tournament selection: Choose two individuals a and b (assume a wins)

$$\begin{aligned} &\text{if } a_i \neq b_i && \text{then} && \\ &\qquad \text{if } a_i = 1 && \text{then} && p_i \leftarrow p_i + \frac{1}{n} \\ & && \text{else} && p_i \leftarrow p_i + \frac{1}{n} \end{aligned}$$

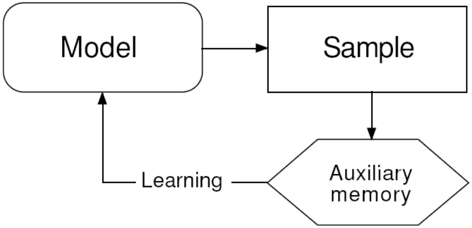
- i.e. the model is updated by (similar to ACO)

GA as MBS

- Bits in the genome were chosen independently. Now model-based: What about schemata?
- In order to capture the essential idea of GA (building blocks!) the probabilistic model must be different from the ACO model (i.e. the pheromone matrix + update)
- Modeling dependencies between string positions e.g.
    - learning a chain distribution as in ACO starting at the first character of the string and setting the next one by a conditional probability
    - by a matrix of pair-wise joint frequencies
    - by a forest of mutually independent dependency trees that represent schemata
    - Bayesian networks

PSO as MBS

- As in GA the “model” is actually a population (which can be represented by a probabilistic model)
- Generate new samples from the individual particles of the previous iteration by random modifications
- Use memory of global, neighborhood or personal best for learning



It is thanks to these eccentrics, whose behaviour is not conform to the one of the other bees, that all fruits sources around the colony are so quickly found.

Karl von Frisch, 1927