Extra Slides for lecture 4: “Constrained and multi-objective optimization & best practises”

These slides combined by TM from:
A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing - slides, Chapters 9 and 14

Multimodal Problems and Spatial Distribution

Chapter 9
Motivation 1: Multimodality

Most interesting problems have more than one locally optimal solution.

Motivation 2: Genetic Drift

- Finite population with global (panmictic) mixing and selection eventually convergence around one optimum
- Often might want to identify several possible peaks
- This can aid global optimisation when sub-optima has the largest basin of attraction
Biological Motivation 1: Speciation

- In nature different species adapt to occupy different environmental niches, which contain finite resources, so the individuals are in competition with each other
- Species only reproduce with other members of the same species (Mating Restriction)
- These forces tend to lead to phenotypic homogeneity within species, but differences between species

Biological Motivation 2: Punctuated Equilibria

- Theory that periods of stasis are interrupted by rapid growth when main population is “invaded” by individuals from previously spatially isolated group of individuals from the same species
- The separated sub-populations (demes) often show local adaptations in response to slight changes in their local environments
Implications for Evolutionary Optimisation

- Two main approaches to diversity maintenance:
  - Implicit approaches:
    - Impose an equivalent of geographical separation
    - Impose an equivalent of speciation
  - Explicit approaches
    - Make similar individuals compete for resources (fitness)
    - Make similar individuals compete with each other for survival

Implicit 1: “Island” Model Parallel EAs

Periodic migration of individual solutions between populations
Island Model EAs contd:

- Run multiple populations in parallel, in some kind of communication structure (usually a ring or a torus).
- After a (usually fixed) number of generations (an *Epoch*), exchange individuals with neighbours.
- Repeat until ending criteria met
- Partially inspired by parallel/clustered systems.

Island Model Parameters 1

- Could use different operators in each island.
- How often to exchange individuals?
  - too quick and all pops converge to same solution
  - too slow and waste time
  - most authors use range ~ 25-150 gens
  - can do it adaptively (stop each pop when no improvement for (say) 25 generations)
Island Model Parameters 2

- How many, which individuals to exchange?
  - usually ~2-5, but depends on population size.
  - more sub populations usually gives better results but there can be a "critical mass" i.e. minimum size of each sub population needed
  - Martin et al found that better to exchange randomly selected individuals than best
  - can select random/worst individuals to replace

Implicit 2: Diffusion Model Parallel EAs

- Impose spatial structure (usually grid) in 1 pop
Diffusion Model EAs

- Consider each individual to exist on a point on a (usually rectangular toroid) grid
- Selection (hence recombination) and replacement happen using concept of a neighbourhood a.k.a. **deme**
- Leads to different parts of grid searching different parts of space, good solutions diffuse across grid over a number of gens

Diffusion Model Example

- Assume rectangular grid so each individual has 8 immediate neighbours
- equivalent of 1 generation is:
  - pick point in pop at random
  - pick one of its neighbours using roulette wheel
  - crossover to produce 1 child, mutate
  - replace individual if fitter
  - circle through population until done
**Implicit 3: Automatic Speciation**

- Either only mate with genotypically/phenotypically similar members or
- Add bits to problem representation
  - that are initially randomly set
  - subject to recombination and mutation
  - when selecting partner for recombination, only pick members with a good match
  - can also use tags to perform fitness sharing (see later) to try and distribute members amongst niches

**Explicit 1: Fitness Sharing**

- Restricts the number of individuals within a given niche by “sharing” their fitness, so as to allocate individuals to niches in proportion to the niche fitness
- need to set the size of the niche $\sigma_{\text{share}}$ in either genotype or phenotype space
- run EA as normal but after each gen set

\[
f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i, j))} \quad sh(d) = \begin{cases} 
1 - d / \sigma & d < \sigma \\
0 & \text{otherwise}
\end{cases}
\]
Explicit 2: Crowding

- Attempts to distribute individuals evenly amongst niches
- Relies on the assumption that offspring will tend to be close to parents
- Uses a distance metric in ph/g genotype space
- Randomly shuffle and pair parents, produce 2 offspring
- 2 parent/offspring tournaments - pair so that \( d(p1,o1) + d(p2,o2) < d(p1,o2) + d(p2,o1) \)

Fitness Sharing vs. Crowding
Multi-Objective Problems (MOPs)

- Wide range of problems can be categorised by the presence of a number of $n$ possibly conflicting objectives:
  - buying a car: speed vs. price vs. reliability
  - engineering design: lightness vs strength
- Two part problem:
  - finding set of good solutions
  - choice of best for particular application

MOPs 1: Conventional approaches

- rely on using a weighting of objective function values to give a single scalar objective function which can then be optimised:

$$f'(x) = \sum_{i=1}^{n} w_i f_i(x)$$

- to find other solutions have to re-optimise with different $w_i$
MOPs 2: Dominance

- we say $x$ dominates $y$ if it is at least as good on all criteria and **better** on at least one

MOPs 3: Advantages of EC approach

- Population-based nature of search means you can *simultaneously* search for set of points approximating Pareto front
- Don’t have to make guesses about which combinations of weights might be useful
- Makes no assumptions about shape of Pareto front - can be convex / discontinuous etc
MOPs 4: Requirements of EC approach

- Way of assigning fitness,
  - usually based on dominance
- Preservation of diverse set of points
  - similarities to multi-modal problems
- Remembering all the non-dominated points you’ve seen
  - usually using elitism or an archive

MOPs 5: Fitness Assignment

- Could use aggregating approach and change weights during evolution
  - no guarantees
- Different parts of pop use different criteria
  - e.g. VEGA, but no guarantee of diversity
- Dominance
  - ranking or depth based
  - fitness related to whole population
MOPs 6: Diversity Maintenance

- Usually done by niching techniques such as:
  - fitness sharing
  - adding amount to fitness based on inverse distance to nearest neighbour (minimisation)
  - (adaptively) dividing search space into boxes and counting occupancy
- All rely on some distance metric in genotype / phenotype space

MOPs 7: Remembering Good Points

- Could just use elitist algorithm
  - e.g. \((\mu + \lambda)\) replacement
- Common to maintain an archive of non-dominated points
  - some algorithms use this as second population that can be in recombination etc
  - others divide archive into regions too e.g. PAES
Working with Evolutionary Algorithms

Chapter 14

Issues considered

- Experiment design
- Algorithm design
- Test problems
- Measurements and statistics
- Some tips and summary
Experimentation

- Has a goal or goals
- Involves algorithm design and implementation
- Needs problem(s) to run the algorithm(s) on
- Amounts to running the algorithm(s) on the problem(s)
- Delivers measurement data, the results
- Is concluded with evaluating the results in the light of the given goal(s)
- Is often documented (see tutorial on paper writing)

Goals for experimentation

- Get a good solution for a given problem
- Show that EC is applicable in a (new) problem domain
- Show that my_EA is better than benchmark_EA
- Show that EAs outperform traditional algorithms (sic!)
- Find best setup for parameters of a given algorithm
- Understand algorithm behavior (e.g. pop dynamics)
- See how an EA scales-up with problem size
- See how performance is influenced by parameters
Example: Production Perspective

- Optimising Internet shopping delivery route
  - Different destinations each day
  - Limited time to run algorithm each day
  - Must always find reasonably good route in limited time

Example: Design Perspective

- Optimising spending on improvements to national road network
  - Total cost: billions of Euro
  - Computing costs negligible
  - Six months to run algorithm on hundreds computers
  - Many runs possible
  - Must produce very good result just once
Perspectives and their goals

- **Design perspective:**
  find a very good solution at least once
- **Production perspective:**
  find a good solution at almost every run
- **Publication perspective:**
  must meet scientific standards (huh?)
- **Application perspective:**
  good enough is good enough (verification!)

These perspectives have very different implications on evaluating the results (yet are often left implicit)

Algorithm design

- Design a representation
- Design a way of mapping a genotype to a phenotype
- Design a way of evaluating an individual
- Design suitable mutation operator(s)
- Design suitable recombination operator(s)
- Decide how to select individuals to be parents
- Decide how to select individuals for the next generation (how to manage the population)
- Decide how to start: initialisation method
- Decide how to stop: termination criterion
Bad example

- I invented “tricky mutation”
- Showed that it is a good idea by:
  - Running standard (?) GA and tricky GA
  - On 10 objective functions from the literature
  - Finding tricky GA better on 7, equal on 1, worse on 2 cases
- I wrote it down in a paper
- And it got published!
- Q: what did I learned from this experience?
- Q: is this good work?

Bad example (cont’d)

- What did I (my readers) did not learn:
  - How relevant are these results (test functions)?
  - What is the scope of claims about the superiority of the tricky GA?
  - Is there a property distinguishing the 7 good and the 2 bad functions?
  - Are my results generalisable?
    - Is the tricky GA applicable for other problems?
    - If so, which ones?
Test problems

- Many experimenters use one or more of:
  - 5 DeJong functions
  - 25 “hard” objective functions
  - Frequently encountered or otherwise important variants of given practical problem
  - Selection from recognized benchmark problem repository (“challenging” by being NP--- ?!)
  - Problem instances made by random generator

- Choice has severe implications on
  - generalizability and
  - scope of the results

Testing on Real Data

- Advantages:
  - Results could be considered as very relevant viewed from the application domain (data supplier)

- Disadvantages
  - Can be over-complicated
  - Can be few available sets of real data
  - May be commercial sensitive – difficult to publish and to allow others to compare
  - Results are hard to generalize
Testing on Standard Data Sets

- Stored in problem repositories, e.g.:
  - OR-Library:  http://www.ms.ic.ac.uk/info.html
  - UCI Machine Learning Repository
    http://www.ics.uci.edu/~mlearn/MLRepository.html

- Advantage:
  - Well-chosen problems and instances (hopefully)
  - Much other work on these → results comparable

- Disadvantage:
  - Not real – might miss crucial aspect
  - Algorithms get tuned for popular test suites

Problem Instance Generators

- Produce simulated data for given parameters,
  - GA/EA Repository of Test Problem Generators
    http://www.cs.uwyo.edu/~wspears/generators.html

- Advantage:
  - Allow very systematic comparisons for they
    - can produce many instances with the same characteristics
    - enable gradual traversing of a range of characteristics (hardness)
  - Can be shared allowing comparisons with other researchers

- Disadvantage
  - Not real – might miss crucial aspect
  - Given generator might have hidden bias
Basic rules of experimentation

- EAs are stochastic →
  - never draw any conclusion from a single run
    - perform sufficient number of independent runs
    - use statistical measures (averages, standard deviations)
    - use statistical tests to assess reliability of conclusions
- EA experimentation is about comparison →
  - always do a fair competition
    - use the same amount of resources for the competitors
    - try different comp. limits (to cope with turtle/hare effect)
    - use the same performance measures

Things to Measure

- Average quality of solution found in given time
- Average time to find a target quality of solution
- Proportion of runs within % of target
- Best result over \( n \) runs
- Amount of computing required to reach target in given time with % confidence
- ...
What time units do we use?

- Elapsed time?
  - Depends on computer, network, etc…
- CPU Time?
  - Depends on skill of programmer, implementation, etc…
- Generations?
  - Difficult to compare when parameters like population size change
- Evaluations?
  - Evaluation time could depend on algorithm, e.g. direct vs. indirect representation

Measures

- Performance measures (off-line)
  - Efficiency (algorithm speed)
    - CPU time
    - No. of steps, i.e., generated points in the search space
  - Effectiveness (algorithm quality)
    - Success rate
    - Solution quality at termination
- “Working” measures (on-line)
  - Population distribution (genotypic)
  - Fitness distribution (phenotypic)
  - Improvements per time unit or per genetic operator
  - …
Performance Metrics

- No. of (distinct?) generated points in the search space
  - = no. of fitness evaluations (don’t use no. of generations)
- AES: average no. of evaluations to solution
- SR: success rate = % of runs finding a solution (individual with acceptable quality / fitness)
- MBF: mean best fitness at termination,
  - i.e., best per run, averaged over a set of runs
- SR ≠ MBF
  - Low SR, high MBF: good approximator (more time helps?)
  - High SR, low MBF: “Murphy” algorithm

Note that perspective will govern choice of metric

Designing fair experiments

- Basic rule: use the same computational limit for each competitor
- Allow each EA the same no. of evaluations, but
  - Beware of hidden labour, e.g. in heuristic mutation operators
  - Beware of possibly fewer evaluations by smart operators
- EA vs. heuristic: allow the same no. of steps:
  - Defining “step” is crucial, might imply bias!
  - Scale-up comparisons eliminate this bias
Example: off-line performance measure evaluation

- Which algorithm is better?
- Why?
- When?

Example: on-line performance measure evaluation

Which algorithm is better?
- Why and when?
Example: averaging on-line measures

Averaging can “choke” interesting information

Example: overlaying on-line measures

Overlay of curves can lead to very “cloudy” figures
Statistical Comparisons and Significance

- Algorithms are stochastic
- Results have element of “luck”
- Sometimes can get away with less rigour – e.g. parameter tuning
- For scientific papers need to show statistical significance of comparisons whenever a claim is made e.g.: “Newbie recombination is better than uniform crossover”,

Example

<table>
<thead>
<tr>
<th>Trial</th>
<th>Old Method</th>
<th>New Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
<td>657</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>543</td>
</tr>
<tr>
<td>3</td>
<td>556</td>
<td>654</td>
</tr>
<tr>
<td>4</td>
<td>573</td>
<td>565</td>
</tr>
<tr>
<td>5</td>
<td>420</td>
<td>654</td>
</tr>
<tr>
<td>6</td>
<td>590</td>
<td>712</td>
</tr>
<tr>
<td>7</td>
<td>700</td>
<td>456</td>
</tr>
<tr>
<td>8</td>
<td>472</td>
<td>564</td>
</tr>
<tr>
<td>9</td>
<td>534</td>
<td>675</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>643</td>
</tr>
</tbody>
</table>

Average: 545.7 | 612.3

Is the new method better?
Example (cont’d)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Old Method</th>
<th>New Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>500</td>
<td>657</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>543</td>
</tr>
<tr>
<td>3</td>
<td>556</td>
<td>654</td>
</tr>
<tr>
<td>4</td>
<td>573</td>
<td>565</td>
</tr>
<tr>
<td>5</td>
<td>420</td>
<td>654</td>
</tr>
<tr>
<td>6</td>
<td>590</td>
<td>712</td>
</tr>
<tr>
<td>7</td>
<td>700</td>
<td>456</td>
</tr>
<tr>
<td>8</td>
<td>472</td>
<td>564</td>
</tr>
<tr>
<td>9</td>
<td>534</td>
<td>675</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>643</td>
</tr>
</tbody>
</table>

Average: 545.7 612.3
SD: 73.5962635 73.5473317
T-test: 0.07080798

- Standard deviations supply additional info
- T-test (and alike) indicate the chance that the values came from the same underlying distribution (difference is due to random effects) E.g. with 7% chance in this example.

Statistical tests for comparisons

- T-test:
  - most widely used,
  - easy to implement & is in all stats packages
  - assumes:
    - Data taken from continuous interval or close approximation
    - Normal distribution
    - Similar variances for too few data points
    - Similar sized groups of data points
- Other tests:
  - Wilcoxon – preferred to t-test where numbers are small or distribution is not known.
  - F-test – tests if two samples have different variances.
Statistical Resources

- Lots of help and stats tutorials available on-line
- http://faculty.vassar.edu/lowry/webtext.html
- Microsoft Excel
- http://www.octave.org/

Better example: problem setting

- I invented myEA for problem X
- Looked and found 3 other EAs and a traditional benchmark heuristic for problem X in the literature
- Asked myself when and why is myEA better
**Better example: experiments**

- Found/made problem instance generator for problem X with 2 parameters:
  - $n$ (problem size)
  - $k$ (some problem specific indicator)
- Selected 5 values for $k$ and 5 values for $n$
- Generated 100 problem instances for all combinations
- Executed all alg’s on each instance 100 times (benchmark was also stochastic)
- Recorded AES, SR, MBF values w/ same comp. limit (AES for benchmark?)
- Put my program code and the instances on the Web

**Better example: evaluation**

- Arranged results “in 3D” ($n,k$) + performance (with special attention to the effect of $n$, as for scale-up)
- Assessed statistical significance of results
- Found the niche for my_EA:
  - Weak in … cases, strong in … cases, comparable otherwise
  - Thereby I answered the “when question”
- Analyzed the specific features and the niches of each algorithm thus answering the “why question”
- Learned a lot about problem X and its solvers
- Achieved generalizable results, or at least claims with well-identified scope based on solid data
- Facilitated reproducing my results → further research
Some tips

- Be organized
- Decide what you want & define appropriate measures
- Choose test problems carefully
- Make an experiment plan (estimate time when possible)
- Perform sufficient number of runs
- Keep all experimental data (never throw away anything)
- Use good statistics ("standard" tools from Web, MS)
- Present results well (figures, graphs, tables, …)
- Watch the scope of your claims
- Aim at generalisable results
- Publish code for reproducibility of results (if applicable)