Practical Issues in CI

Ideas taken from:

Luigi Barone  Jim Smith
Gusz Eiben  Moshe Sipper
Daniele Loiacono  Paul Weigand
Cara MacNish  Lyndon While
Elena Popovic  Xin Yao
Mike Preuss
Global optimisation: a search through a set of solutions $S$ for the solution $x^*$ that is best according to some measure of fitness.

Deterministic approaches can guarantee to find $x^*$, but they often need super-polynomial time.

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
Heuristic Approaches for Optimisation

- **Heuristic approaches** (generate and test) use rules to decide which $x \in S$ to generate next.
- However, they do not guarantee that the returned solutions are globally optimal.
- Heuristics utilising a neighbourhood structure (e.g. hill-climbers) may guarantee that the locally optimal solution is found.
  - Often very quick to identify good solutions.
- **Need to balance exploration versus exploitation**
  - Exploration: search for novel good solutions.
  - Exploitation: use known good solutions to generate new solutions.
Diversity is a measure of the proportion of the search landscape covered by the population.

Diversity preservation schemes attempt to ensure a consistent coverage of the search landscape.

Genetic drift occurs when the population wanders aimlessly with little selection pressure.
Anytime behaviour: the algorithm is always able to provide a solution to the problem – the solution is refined over time.

The convergence rate is the rate at which the search converges to a (potentially local optimum) final best solution.
Early phase: quasi-random population distribution

Mid phase: population situated on/around “hills” of the search landscape

Late phase: population converges to optima

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
Fitness Progression

- Are long runs beneficial?
  - it depends how much you want the last bit of progress
  - it may be better to do several shorter runs

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
Is it worth expending effort on “smart” initialisation?
- possibly, if good solutions/methods exist
- but care is needed to ensure no bias is induced
CI as General Purpose Problem Solvers

• CI approaches are considered general-purpose problem solvers

• For most problems, a problem-specific tool will perform “better” than a generic search algorithm on most instances
  – better in terms of final performance and/or efficiency of the search
  – but has limited utility
  – and does not perform well on all problem instances

• Goal for CI is to provide robust tools that provide:
  – evenly good performance
  – over a range of problems and instances
CI as General Purpose Problem Solvers

Tailored approach provides superior performance over a limited range of problem instances

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
The Importance of Domain Knowledge

- **Domain knowledge** is “information” about the problem that can be used to guide the search
- Domain knowledge can be exploited by the algorithm to improve the efficiency of the search
- Domain knowledge can be a good or bad thing
  - too little domain knowledge may make the search space unnecessarily big and hard to navigate
  - too much domain knowledge may exclude novel or surprising solutions or induce a bias in the search
In general, the use of domain knowledge improves performance, at the cost of generality.
No Free Lunch Theorem

- Early view thought CI approaches were superior to other approaches across all problem types – not correct!
- The no free lunch theorem states that “any two algorithms are equivalent when their performance is averaged across all possible problems”
- In other words, need to apply the right algorithm for each particular problem type – matching algorithms to problems gives higher average performance than does applying a fixed algorithm to all
- Understanding which algorithms are applicable to which problem “classes” is an open problem
Niching and Speciation

- Most interesting problems have more than one locally-optimal solution (multi-modal)

- Often interested in identifying several optima
  - can aid global optimisation when a sub-optimal solution has the largest basin of attraction

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
Niching (or speciation) refers to the formation of separate distinct groups (niches) in the population. Solutions within a niche are similar to each other, while solutions from different niches are different to each other. Can be used to locate the multiple local optima. Helps encourage and maintain diversity, and thus explore the search space better.

http://www.cs.vu.nl/~gusz/ecbook/ecbook-course.html
Niching and Speciation

- Implicit approaches to niching:
  1. impose an equivalent of geographical separation
  2. impose an equivalent of speciation at the genetic level

- Explicit approaches to niching:
  1. fitness sharing: make similar individuals compete for resources, distributing individuals to niches in proportion to the niche fitness
  2. crowding: make similar individuals compete for survival, distributing individuals evenly amongst niches

- All techniques rely on some distance metric
- Diversity maintenance schemes use similar approaches
Co-evolution

- **Co-evolutionary algorithm**: an algorithm in which fitness evaluation is based on interactions between individuals rather than a separate fitness metric.
  - changing the set of individuals used for evaluation can affect the fitness ranking of individuals

- **Two main forms:**
  1. **cooperative co-evolution**: decomposes a problem into simpler sub-tasks (layered learning), combining resultant sub-solutions to form a solution to the original problem.
  2. **competitive co-evolution**: aims for an “arms race” between competitors in order to drive improvement through parallel adaptive changes in strategies.
**Memetic Algorithms**

- **Memetic algorithm**: the combination of a CI technique with separate individual learning capable of performing local refinements
  - analogous to the evolutionary transfer of cultural ideas through memes

- Synonymous with Baldwinian EAs, Lamarckian EAs, cultural algorithms, or genetic local search

- Attempts to combine the best features of both schemes:
  - the global exploration power of the CI technique
  - the local exploitation power of local search
Multi-objective Optimisation

A traditional optimisation problem is one where solution performance is measured by a single metric (fitness)
- the metric may be a combination of many sub-factors

A multi-objective optimisation problem (MOOP) is one where solution performance is measured by more than one metric
- e.g. speed vs. safety vs. price for car designs

Fitness is hence a vector, not a scalar

Complicates the algorithm’s selection process
- there is no total ordering on fitnesses, only a partial ordering
Multi-objective Optimisation

- An algorithm for solving a MOOP returns a set of solutions offering varying trade-offs between the different objectives

- Population-based approaches simultaneously explore the search space for this set
  - a parallelised search of different combinations of weights
  - does not need \textit{a priori} information about which combinations are interesting
  - makes no assumptions about the shape of the trade-off set
The rank of a solution X is the number of solutions that dominate X. Solution X dominates Y if X is better or equal to Y in every objective and better in at least one.

Consider a MOOP with two objectives, both to be maximised.

Selection is based primarily on ranks, although some other measure is needed to break ties.
Constraints

- **Constraint**: impose a *requirement* on a solution instead of a measure of goodness
  - commonly cast in terms of equality or inequality functions
- Feasible space: the set of (valid or **feasible**) solutions that observe all problem constraints
- Infeasible space: the set of (invalid or **infeasible**) solutions that violate one or more constraints
- Denote the whole search space as $S$ and the feasible space as $F$, $F \subset S$
- Note that the global optimum in $F$ may not be the same as that in $S$
Repair is the process of converting an infeasible solution into a feasible one.
There are many constraint-handling techniques:

1. **penalty function**: introduce a penalty to the objective function that quantifies the constraint violation
2. **repair**: map (repair) an infeasible solution into a feasible one
3. **purist approach**: reject all infeasible solutions in the search
4. **separatist approach**: consider the objective function and constraints separately
5. **MOOP approach**: add constraint violation as a separate objective in a MOOP
6. **hybrid approach**: mix two or more different constraint handling techniques
A noisy fitness function results from applying a noise function to a underlying fitness function.

For each point, the noisy fitness function returns the true fitness value perturbed by noise.
Noise

- With noise, an algorithm cannot determine true performance, only an estimate
- Noise affects the selection process:
  1. a superior solution may be erroneously judged to be inferior, causing it to be discarded
  2. an inferior solution may be erroneously judged to be superior, causing it to survive and influence the search
- These behaviours cause undesirable effects:
  - the learning rate is reduced
  - the system does not retain what it has learnt
  - exploitation is limited
  - fitness does not improve monotonically with time
The standard approach is to resample the fitness numerous times and average the results. But this reduces the efficiency of the algorithm. And how many times is sufficient?

- Averaging until some measure of the standard error is lower than a threshold is not always best.

Other approaches attempt to either bound the effects of the error or compensate for it by changing how the search proceeds.

One key observation: you don’t need to re-evaluate to accurately determine true fitness, but only to ensure the correct selection of “good” solutions.
Dynamic Problems

- Some problems involve a fitness function that changes over time
  - e.g. due to temporal effects or environmental adaptation
- The algorithm must adapt to this change
- Requires online learning
- Population momentum can be a good thing or a bad thing

http://www.natural-selection.com
Expensive Fitness Functions

- Usually, the cost of a fitness evaluation is the limiting performance factor in CI optimisation
  - we typically ignore the cost associated with maintaining the search (the running cost of the algorithm)

- What if a fitness evaluation is prohibitively expensive?
  - e.g. a discrete-event simulation lasting many hours

- The high cost of a fitness evaluation limits the number of solutions that can be trialed

- The emphasis of the search algorithm is hence on improvement, not optimality or even near-optimality
  - exploration is forgone for exploitation
Offline vs. Online Learning

- **Offline learning**: the system learns before use from a series of pre-defined training instances
  - strategy fixed once the initial training phase is completed
  - needs a comprehensive set of training data
  - not always feasible, or sometimes prohibitively slow

- **Online learning**: the system learns while in use from each training instance encountered
  - continues to adapt while being used
  - decisions made from an incomplete set of training data
  - typically has a limited time to learn between actions
  - needs to balance **exploitation** of known solutions vs. **exploration** of novel good solutions
Learning Paradigms

- **Supervised learning**: training by comparison with known *a priori* input/output examples (training set)
  - requires annotated training instances (teaching signal)
  - offers a direct and concrete form of feedback

- **Unsupervised learning**: offline training, without the use of a training set
  - no “detailed” question
  - the system discovers patterns in the input
  - less direct and more noisy
  - but may be the only way possible if no teaching signal is available
  - more powerful
Unsupervised Learning

Question: tell me about this data
Supervised Learning

- Question: what is an apple?
Reinforcement Learning

- **Reinforcement learning**: online training using feedback from the environment (reward) to assess the quality of actions
  - attempts to learn a policy that maps states to actions that maximises long-term reward for the agent
  - differs from supervised learning in that an explicit teaching signal is not used, nor are sub-optimal actions explicitly corrected

![Diagram showing the interaction between Agent and Environment](image)
Experimental Methodology

- CI algorithms are **stochastic**, thus 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

- Experimentation is about comparison, thus always perform a fair competition between the different algorithms
  - use the same amount of resources for the competitors
  - try different competition limits
  - use the same performance measures
Performance Measures

- Offline performance measures:
  1. effectiveness (algorithm quality)
     - success rate: the percentage of runs that find a solution with acceptable quality/fitness
     - mean best fitness at termination
  2. efficiency (algorithm speed)
     - CPU time to completion
     - number of solutions (fitness evaluations) trialed

- Online (during run) performance measures:
  1. genotypic population distribution
  2. phenotypic fitness distribution
  3. improvement per time unit or per genetic operator
Parameter Tuning

- **Parameter tuning**: the traditional way of testing and comparing different values before the “real” run
- **Difficulties:**
  - user mistakes in settings can cause errors or sub-optimal performance
  - requires a significant amount of time to determine good values
  - parameters interact: exhaustive search is not practicable
  - good values may become bad during the run
Parameter Control

- **Parameter control**: setting values online, during the actual run

- **Examples**:
  - predetermined time-varying schedule
  - use feedback from the search process
  - encode parameters in genotype and use the optimisation process to determine best settings (self-adaptation)

- **Problems**:
  - finding the optimal (temporal) schedule is not easy
  - optimisation of parameters is indirect (subject to “secondary” selection pressure)
  - still requires user-defined feedback mechanism
Parameter Control

Three major types of parameter control:

1. **deterministic**: some rule modifies parameters without feedback from the search (e.g. based on time)
2. **adaptive**: feedback rule based on some measure monitoring search progress
3. **self-adaptive**: parameter values optimised along with actual solutions by encoding parameters into the genotype of the solution
Applying CI Techniques to a Problem

Things to consider when applying CI to a problem:

1. choose an appropriate representation
2. consider and experiment with the nature of the search landscape
3. quantify fitness and constraint functions carefully
4. consider if a hybridised approach is beneficial
5. determine an appropriate learning/training methodology
6. consider how much domain knowledge to apply
7. choose appropriate operators
8. ensure diversity is maintained
9. minimise feasibility issues
10. determine a strategy for setting/updating control parameters
Questions?