Applying Automated Optimisation Techniques to HPC Applications

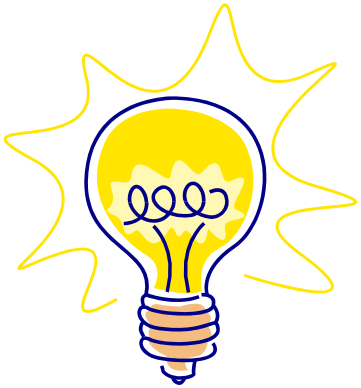
T D Edwards
Cray Centre of Excellence for HECToR
Cray UK Ltd, Edinburgh UK

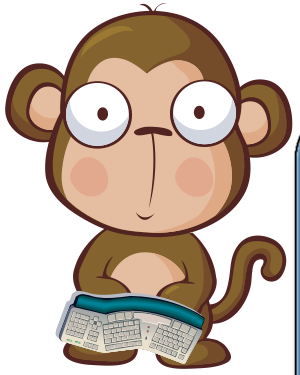
tedwards@cray.com

Background

The Optimisation Cycle

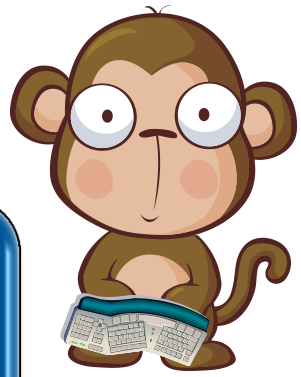
Profile





Modify

Analyse



Levers - Application Parameters

Applications potentially contain lots of helpful tuneable parameters, e.g.

```
Inputfile1.cfg
....
# cache_blk_size – controls the size of blocks
# of work for optimising for cache
cache_blk_size = 7
...
# message_size – Tries to coalesce messages
# into groups of “message_size”
message_size = 1024
...
```

Levers – Compiler

The process of translating source-code to binaries is long and complicated

Compilers (or more accurately compile-writers) do their best to produce something faithful to the source, with good performance.

Usually makes lots of conservative assumptions to cover any possibility.

Users can change these assumptions by passing options to the compiler (risking problems later on)

Most options applicable on a per-file basis, allows lots of flexibility

Most users opt for only a single set of flags for the entire application

Constraints - Numerical Stability / Validity

Floating point arithmetic is an imperfect representation of real-numbers.

Not all numerical methods are perfectly stable. Certain algorithms or lines of source code can be sensitive to small differences in floating point values.

Higher levels of compiler optimisation are more likely to make assumptions that are potentially disruptive (all compilers including CCE make the highest levels of optimisation optional).

The only way to ensure validity is to run the application, potentially for a large number of iterations.

Constraints - Bit Reproducibility

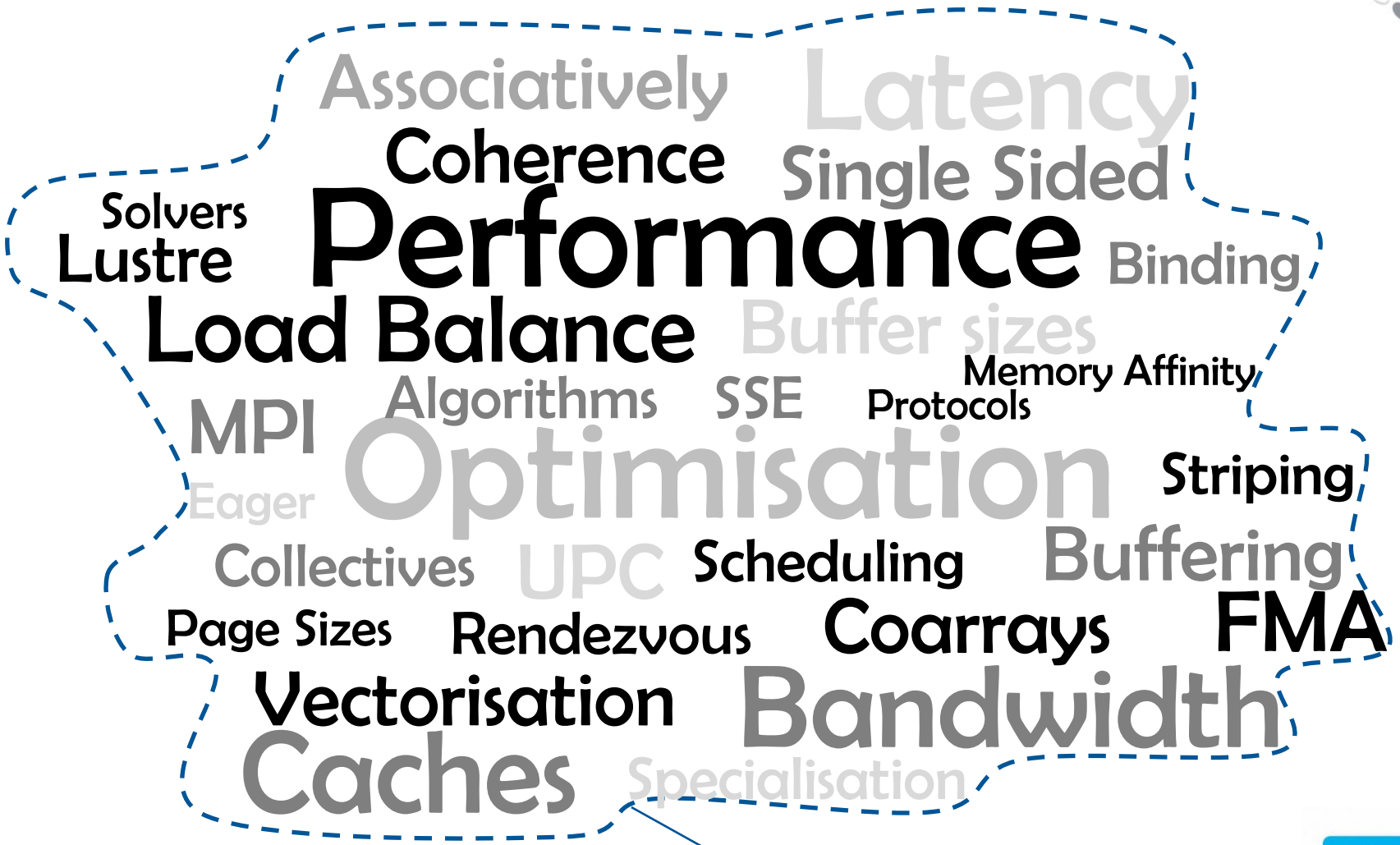
Some applications require that when
- running with the same binary
- running on the same processor architecture
- running from the same input starting data
- yet running on different parallel decompositions

some or all results should be identical (at the bit level or printing level)

The application has to be specially written to even attempt this:
- Has to use specially designed collectives / global sums
- Floating point arithmetic has to be carefully considered.

Problems achieving bit reproducibility can potentially occur at any level; from the optimisations used be the compiler to algorithms in the communications layers.

Levers vs Constraints



Constraints

- Bit-reproducibility
- Numeric Stability
- Model Validity

Upgrading from Magny-Cours to Interlagos

Systems evolve over their lifetime; processors improve, cores increase.
In 2011Q4 HECToR was upgraded from Magny-Cours to Interlagos

	Magny-Cours	Interlagos
Clock Speed	2.1 GHz	2.3 GHz
Cores per node	24	32
D1 Cache	64 kB 2-Way (per core)	16 kB 4-Way (per core)
L2 Cache	512kB (per core)	2048 kB (per module)
L3 Cache	6144 kB (per socket)	8192 kB (per socket)
Floating-point units	1 x SIMD Add-subtract 1 x SIMD Multiply	2 x SIMD FMA

Applications that have been optimised for Magny-Cours may benefit from being re-evaluated on Interlagos
Applications that are new to Cray systems may also benefit from additional tuning

Automating the Optimisation

Optimisation is time consuming and expensive...

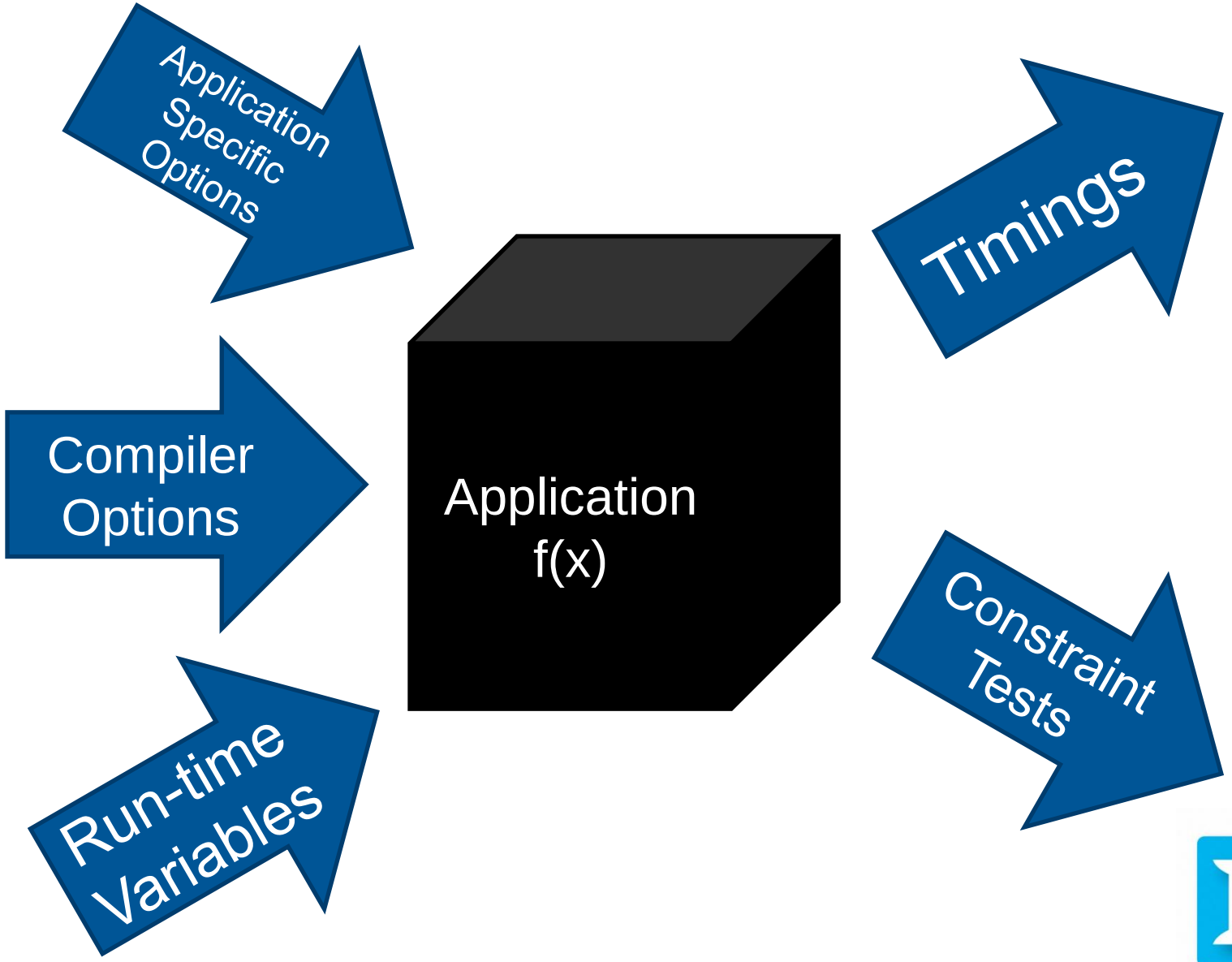
What can be done to automate the process?

In many cases we can optimise by adjusting the external levers (parameters)

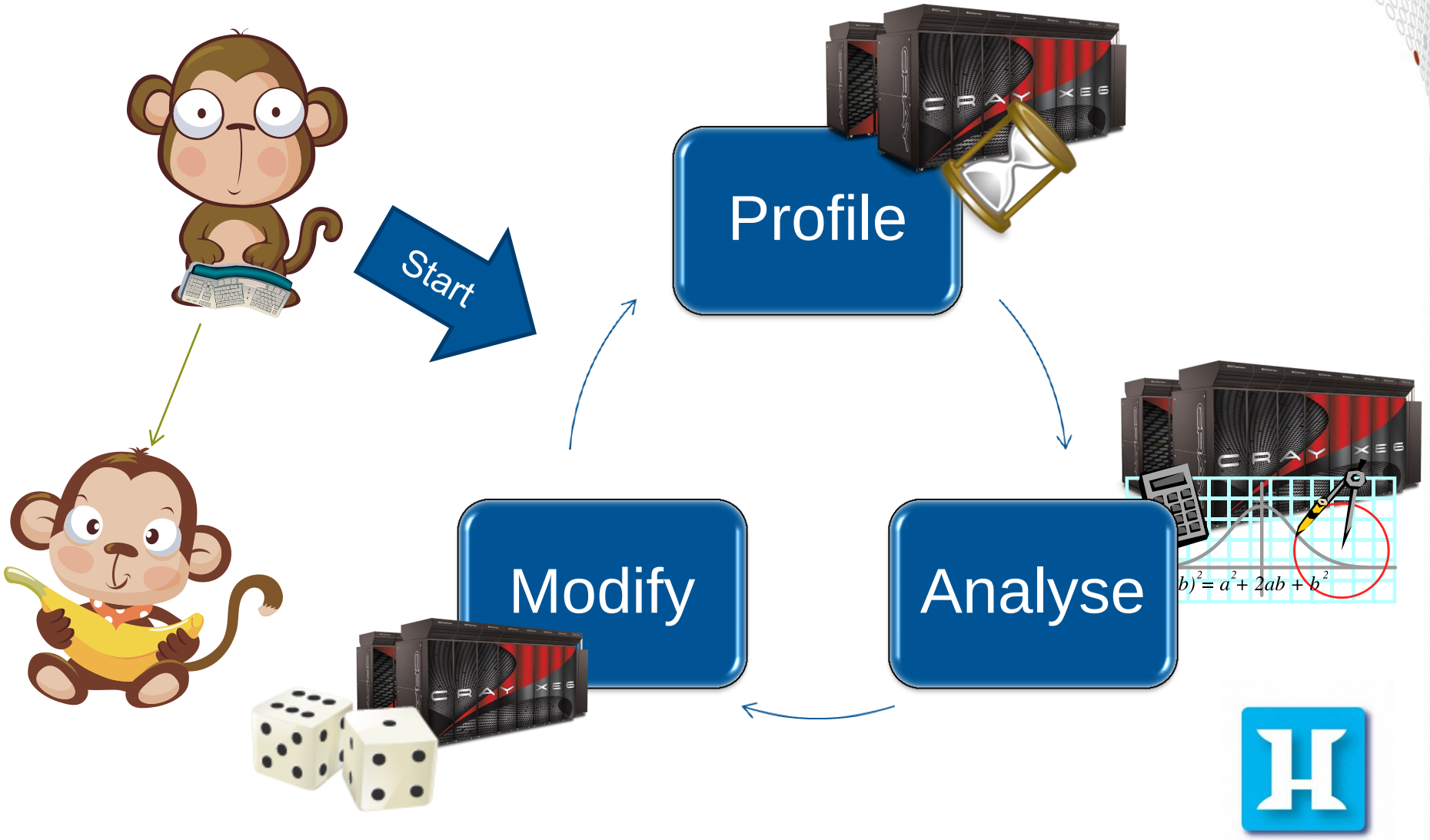
We can treat the application as a “black-box”, where the performance is the evaluation of f(x), where x are the input parameters

We can then apply a search algorithm, that optimises f(x) within the constraints of the application.

Leaves programmers free to look into areas that are not easily optimised automatically.


Application Specific Options
Compiler Options
Run-time Variables
Application f(x)
Timings
Constraint Tests

Automated Optimisation Cycle



Search Algorithms

What kind of algorithms can we use?

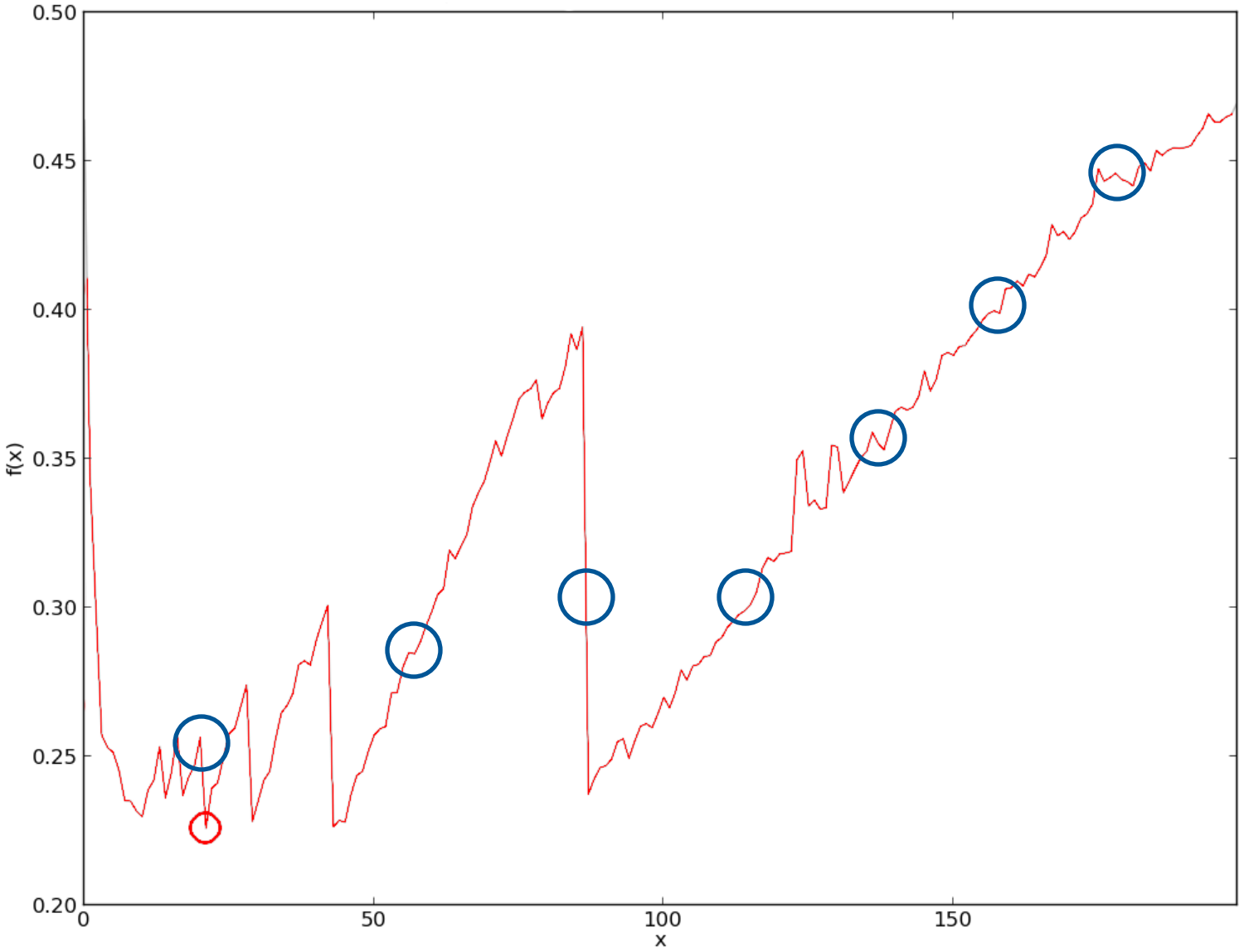
We assume that we know nothing about the analytic properties of the parameter space. There are no guarantees about differentiability or even continuity.

This rules out analytical optimisation techniques and leaves stochastic or meta-heuristic searches.

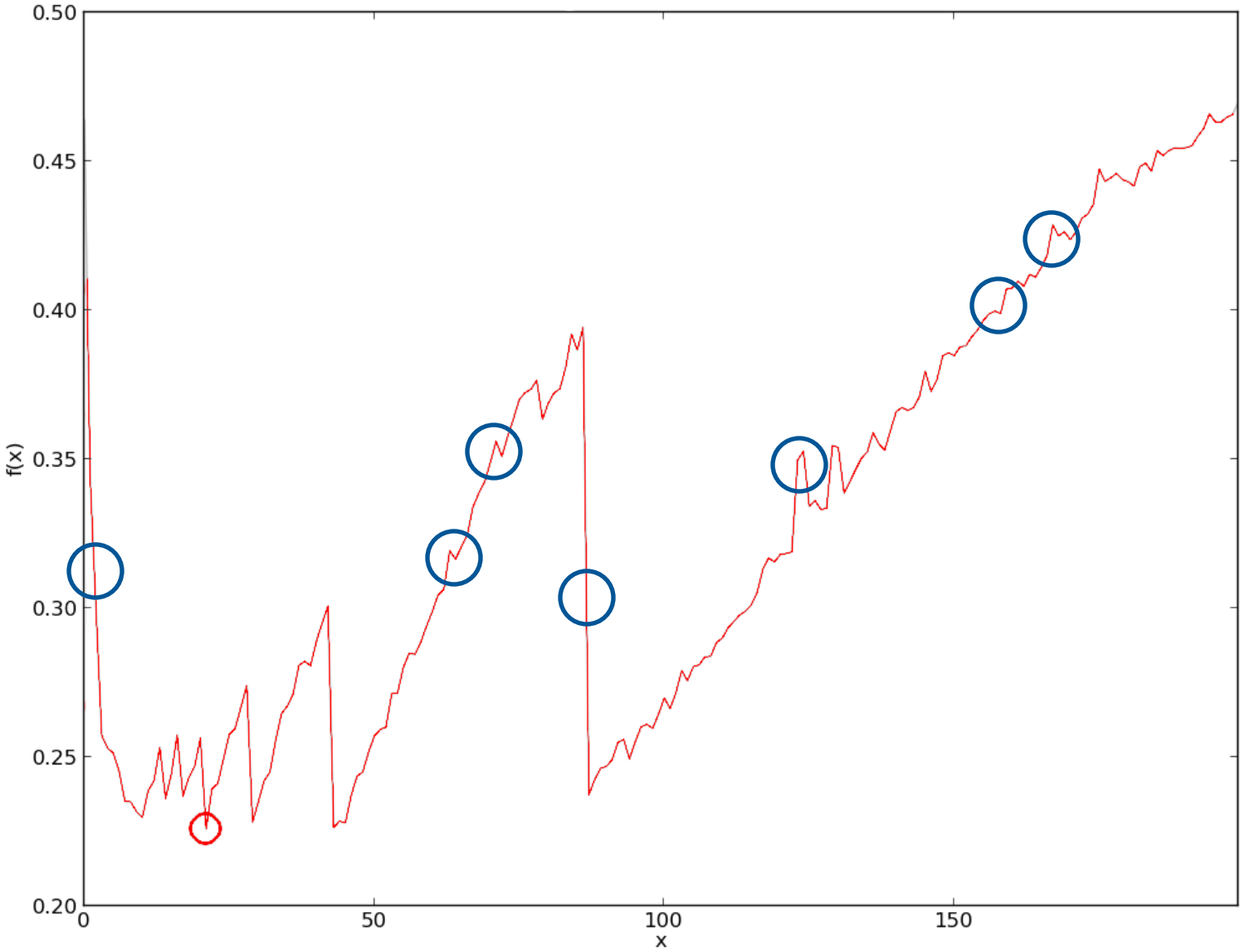
Examples covered here are:

- Interval Search
- Random Search
- Simulated Annealing
- Particle Swarm
- Curve Fitting Search

Interval Search



Random Search



Simulated Annealing

Inspired by the forming of crystals through cooling

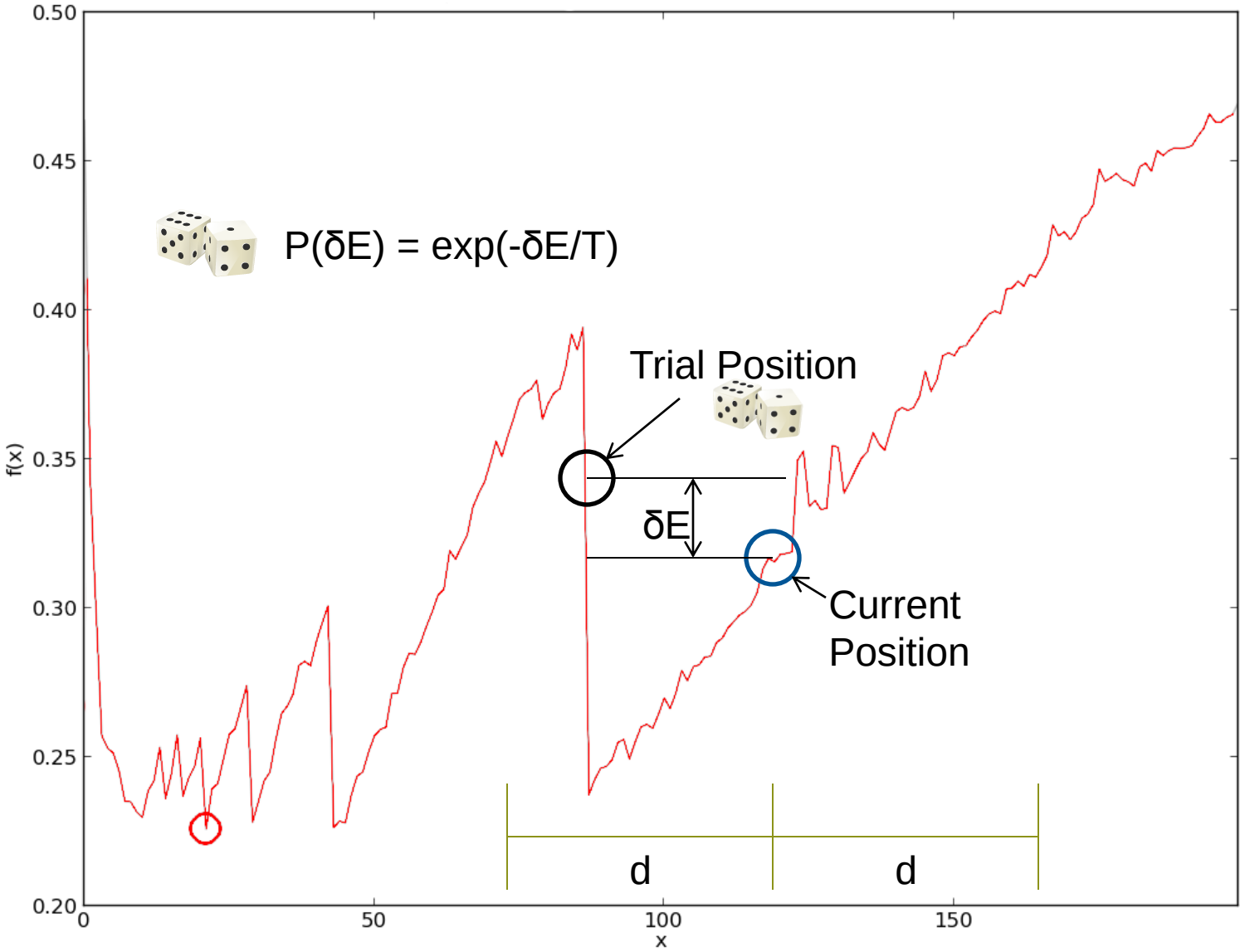
Simulation has a “current” state. Each new state is selected at random (within some distance d of the current state)

The value of f(x) is calculated for the new state (i.e. the model is run with the new parameters). The difference between the two states, if the energy (time) is less then the algorithm moves immediately to the new state. If it is greater, then the model will move according to the Metropolis probability function.

$$P(\delta E) = \exp(-\delta E/T)$$

The temperature, T, will change with each iteration, exponentially decaying with halflife λ, from the maximum T_{max}

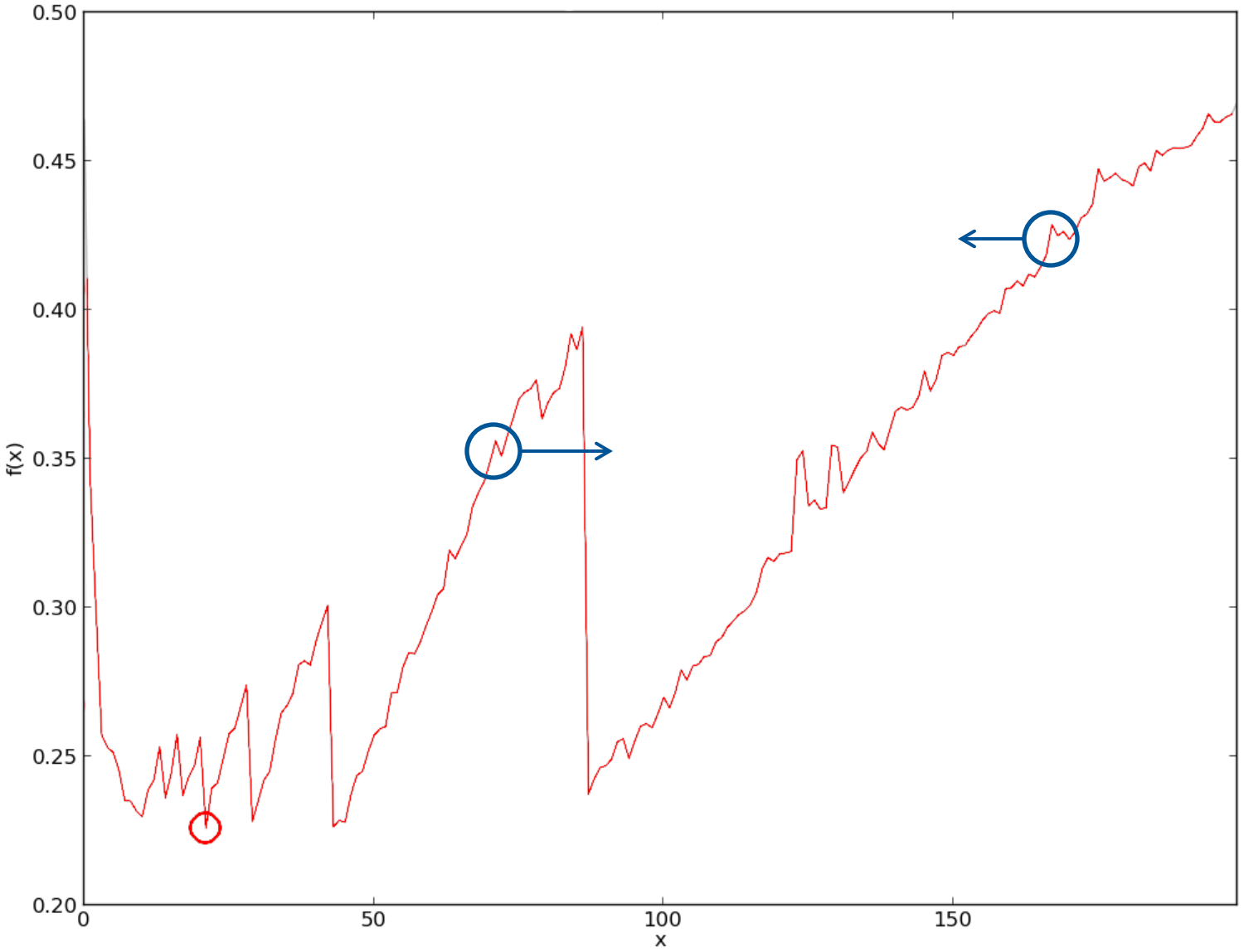
Simulated Annealing



Particle Swarm

- **Inspired by the movement of flocks and swarms in nature**
- **Particles are moving over the parameter space**
  - On the *i*th iteration they are at position x_i with velocity v_i
  - Initial velocity and position are generated randomly
  - Each particle remembers the position of local minimum it has encountered and the position of the current global minimum
- **Each time step one particle is deflected by a random amount towards its local and the global minimums.**
- **The particle is then moved forward in time by one Euler step and the f(x) evaluated at the new position.**

Particle Swarm



Curve Fitting Search

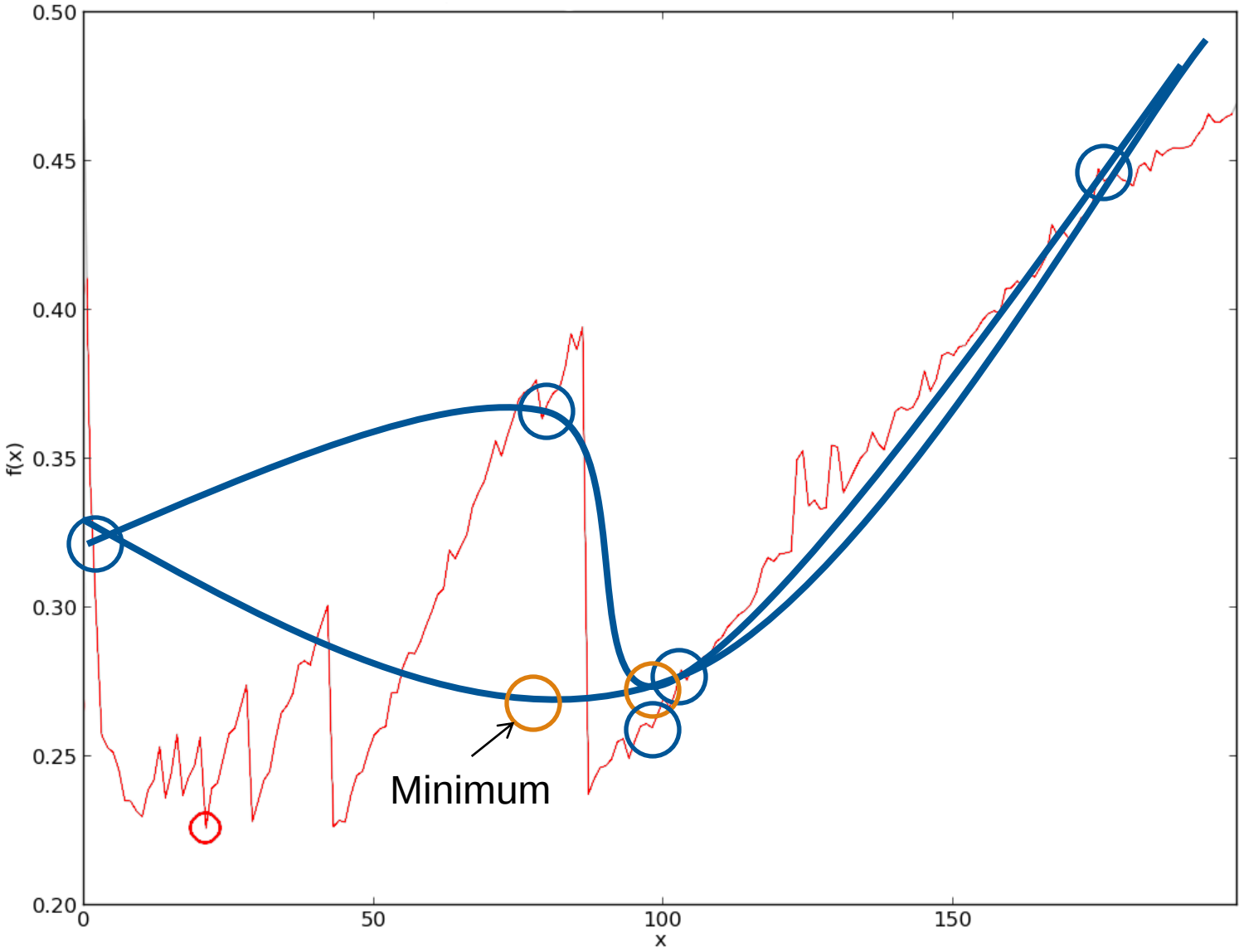
Curve fitting attempts to use information from all previous iterations to identify the global minimum of the parameter space.

Though we cannot guarantee that a space is differentiable or continuous, in most case it will have some nice properties.

This algorithm initially chooses three random points, then uses a multiquadric “Radial Basis Function” to interpolate between the points. This analytic function is then quickly optimised to find the global minimum. This predicted minimum is then evaluated and the process repeated.

If the algorithm has already evaluated the predicted minimum point, then a new point is evaluated at random in an attempt to avoid being trapped in local minima.

Curve Fitting Search



Comparing Algorithms

Evaluation Methodology

Which algorithm is a best general purpose approach for optimisation?

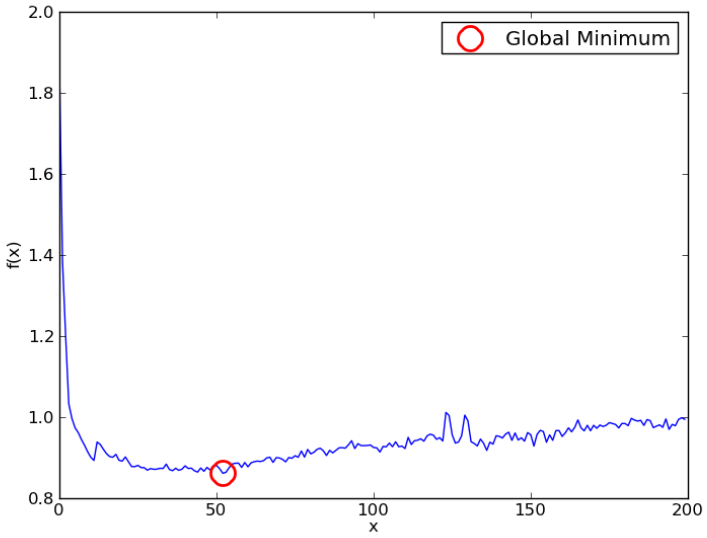
Using real-world datasets from exhaustively searched parameters we can compare algorithms.

In real life we will have no concept of convergence, it is impossible to identify whether a minimum is local or global.

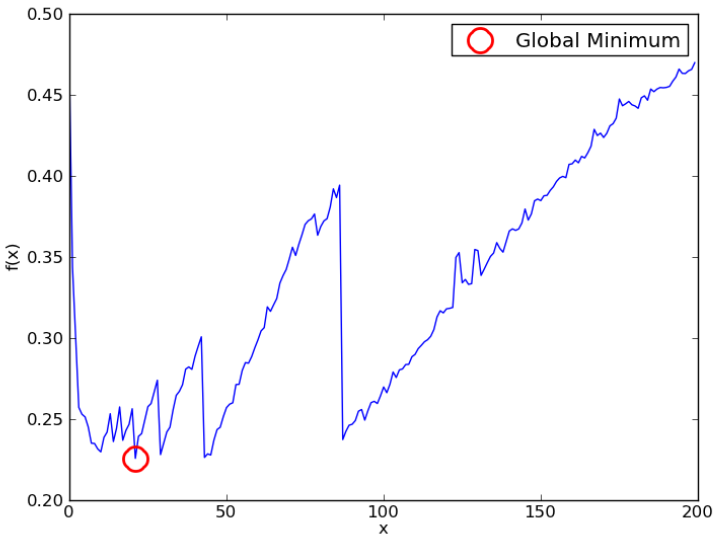
Therefore we cannot compare algorithms on the number of iterations before finding the minimum

Instead, we use the number of iterations as the limit, we can compare algorithms on how close to the minimum then got in a fixed number.

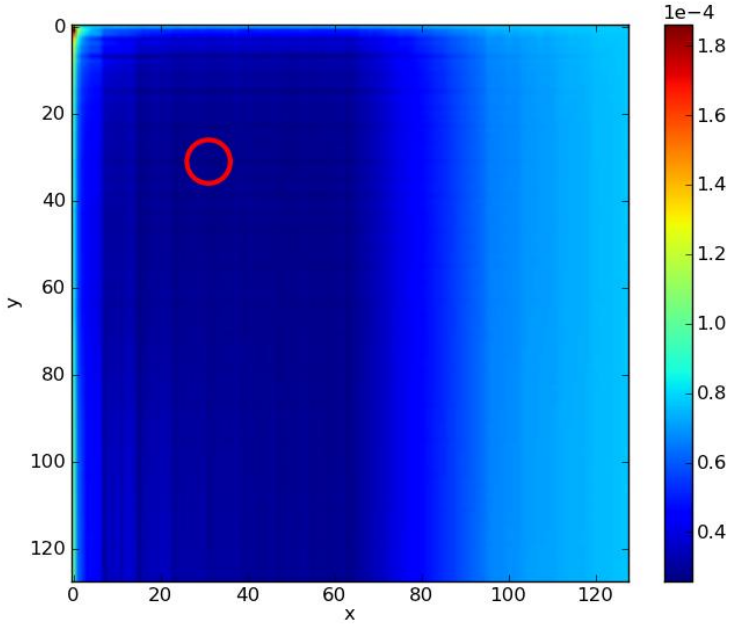
1D Smooth and Shallow Dataset



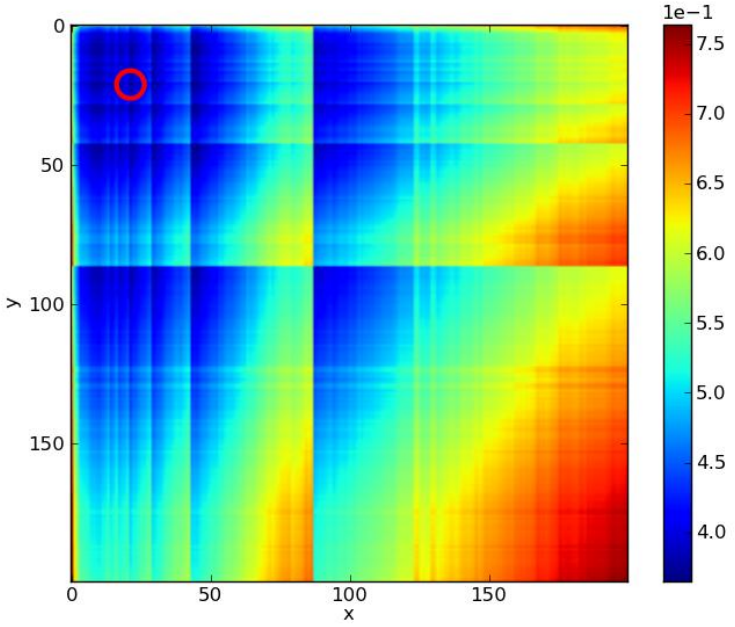
1D Sharp and Steep Dataset



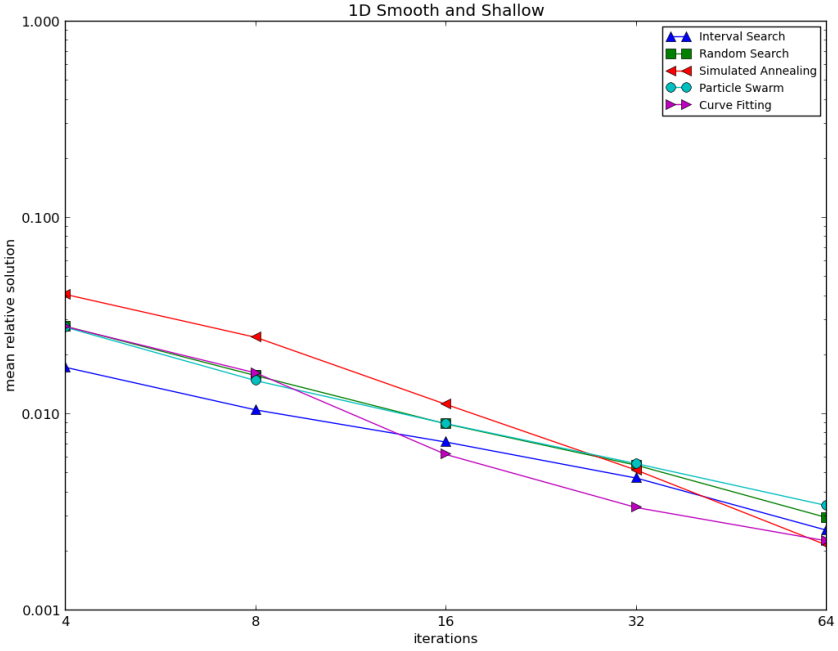
2D Smooth and Shallow Dataset

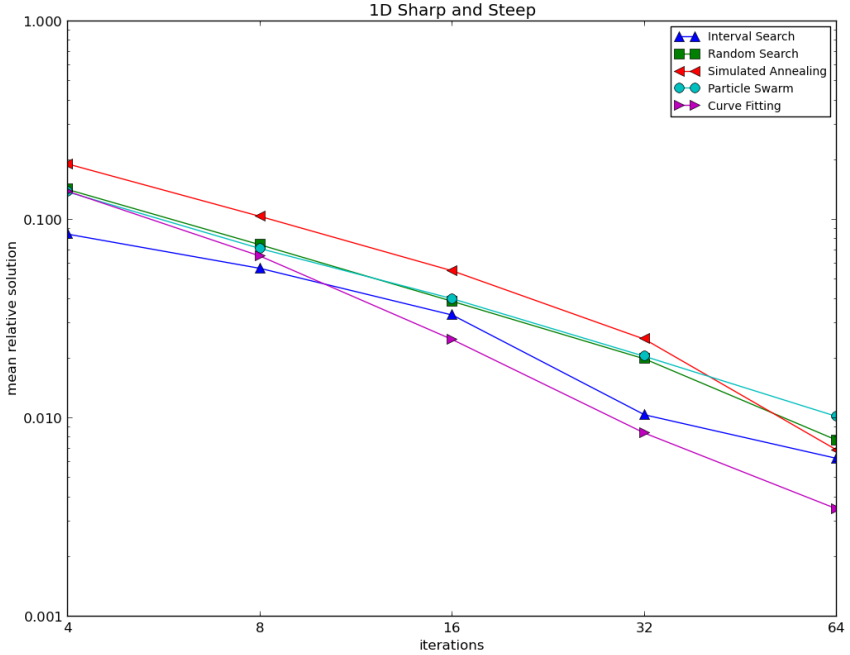


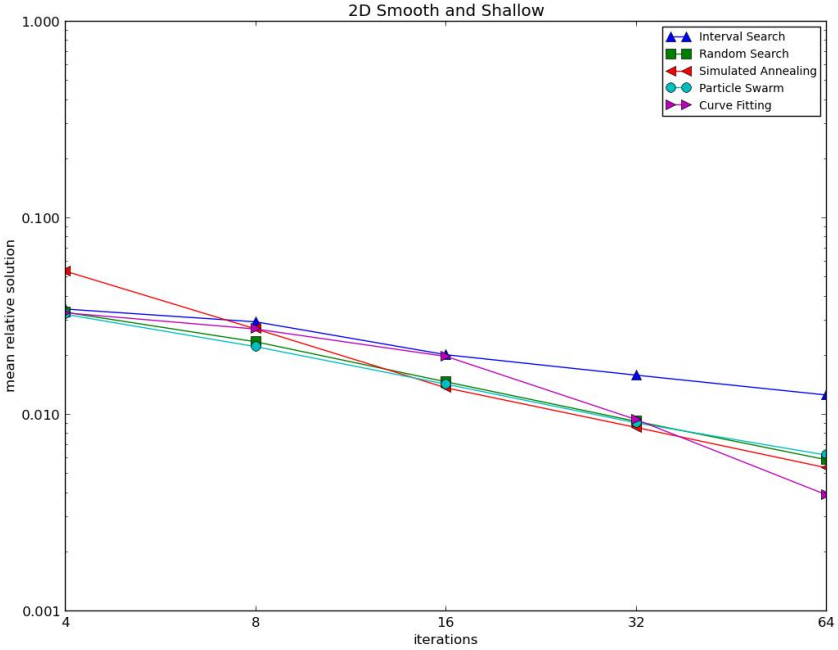
2D Sharp and Steep Dataset

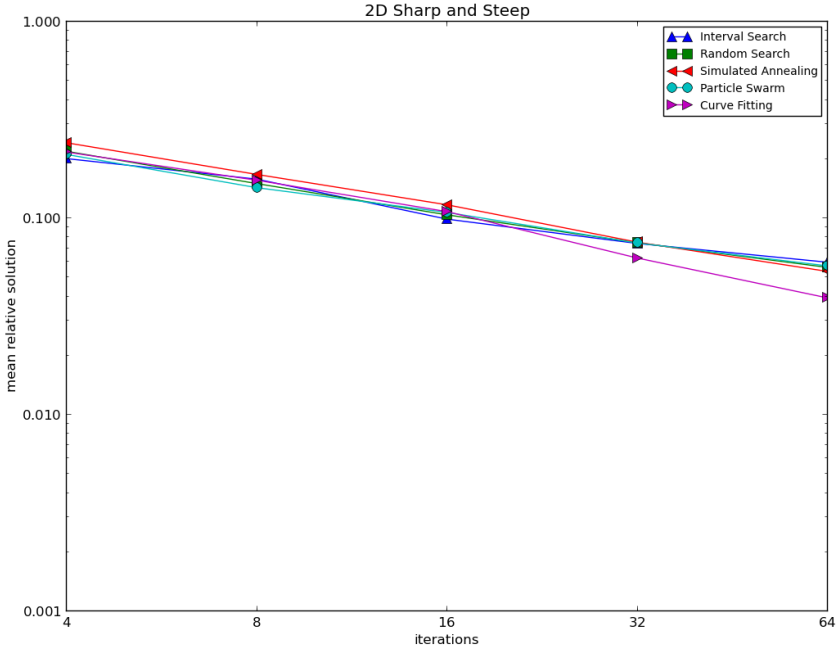


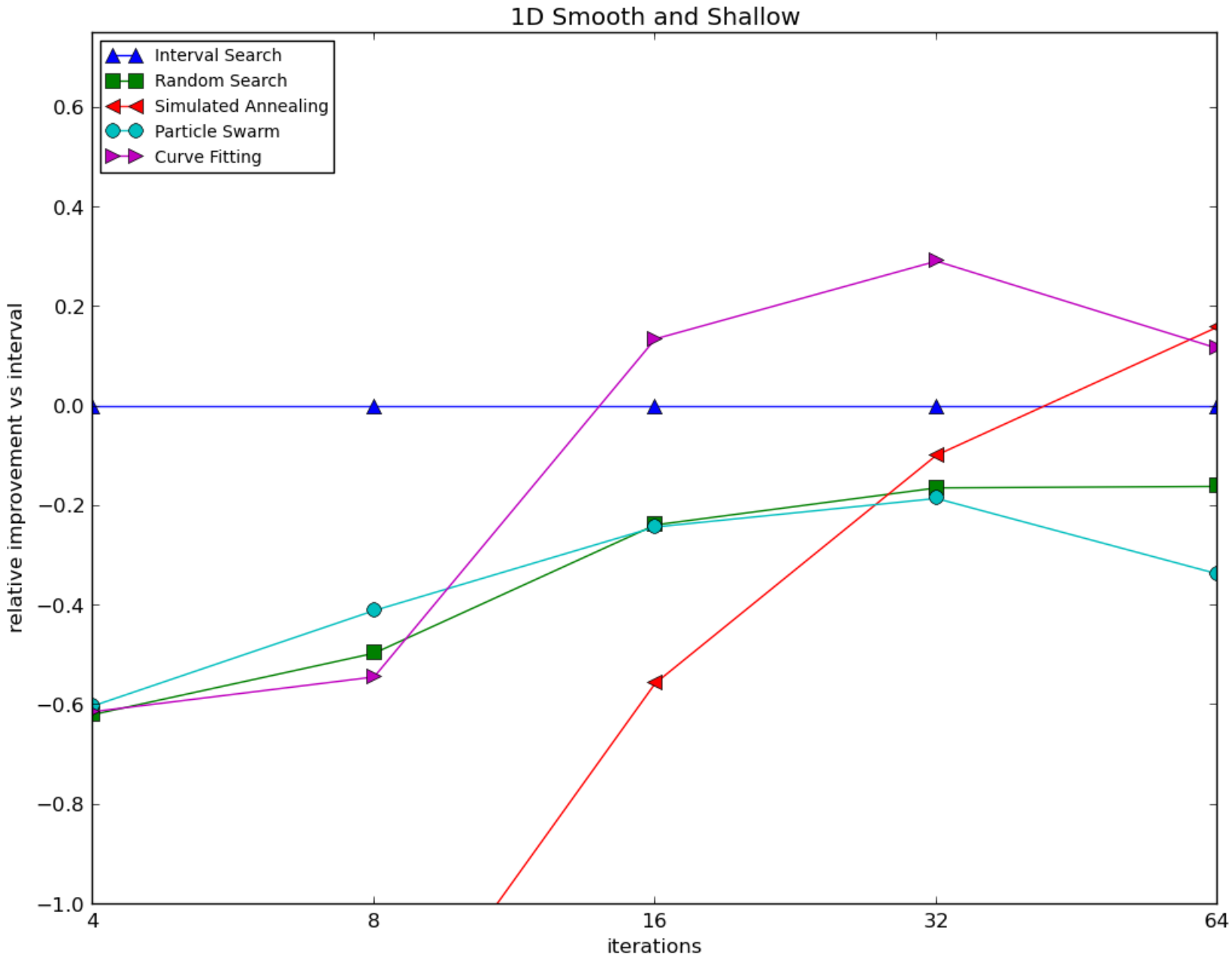
Results

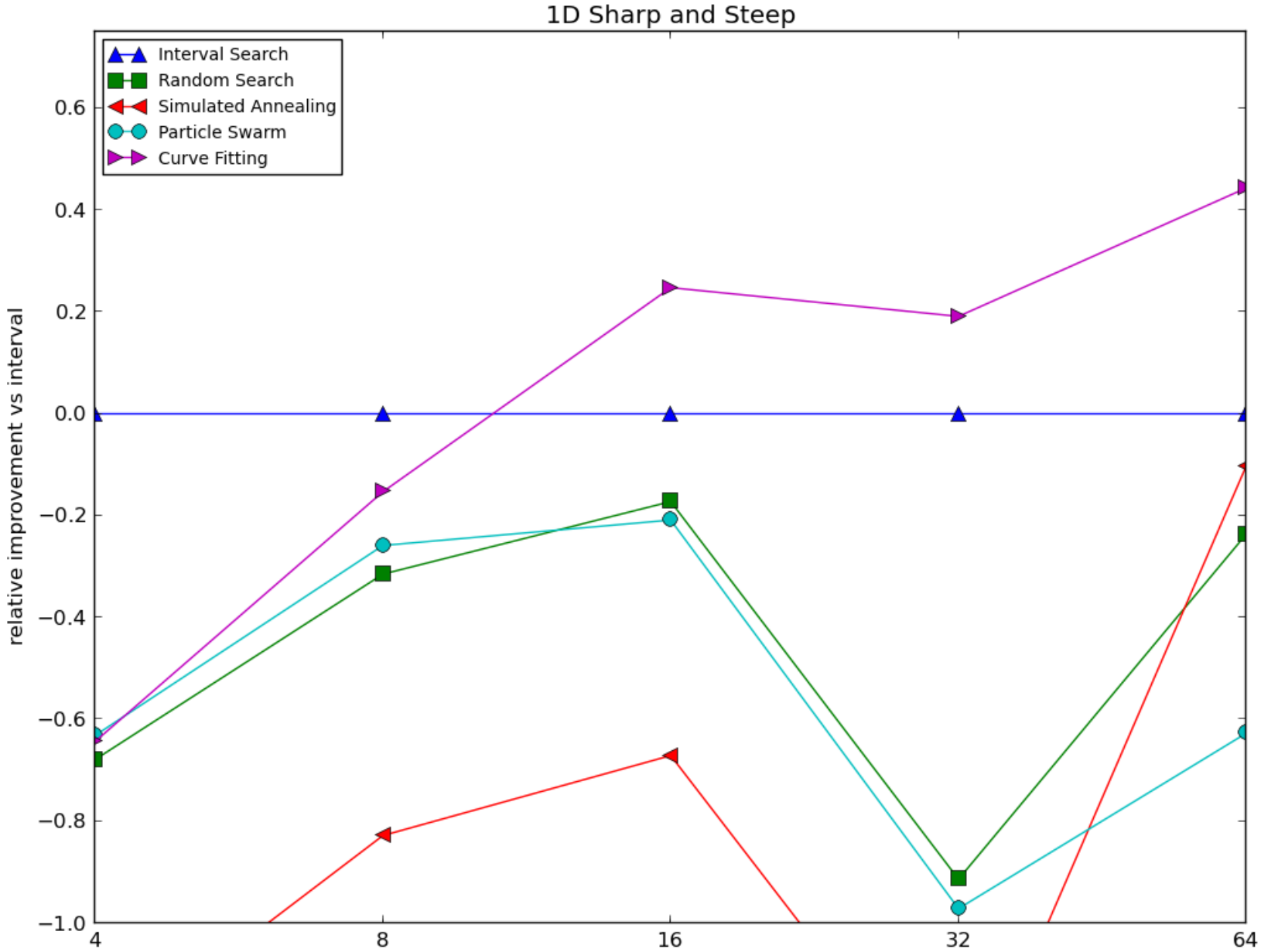

1D Smooth and Shallow
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
mean relative solution
1.000
0.100
0.010
0.001
4
8
16
32
64
iterations

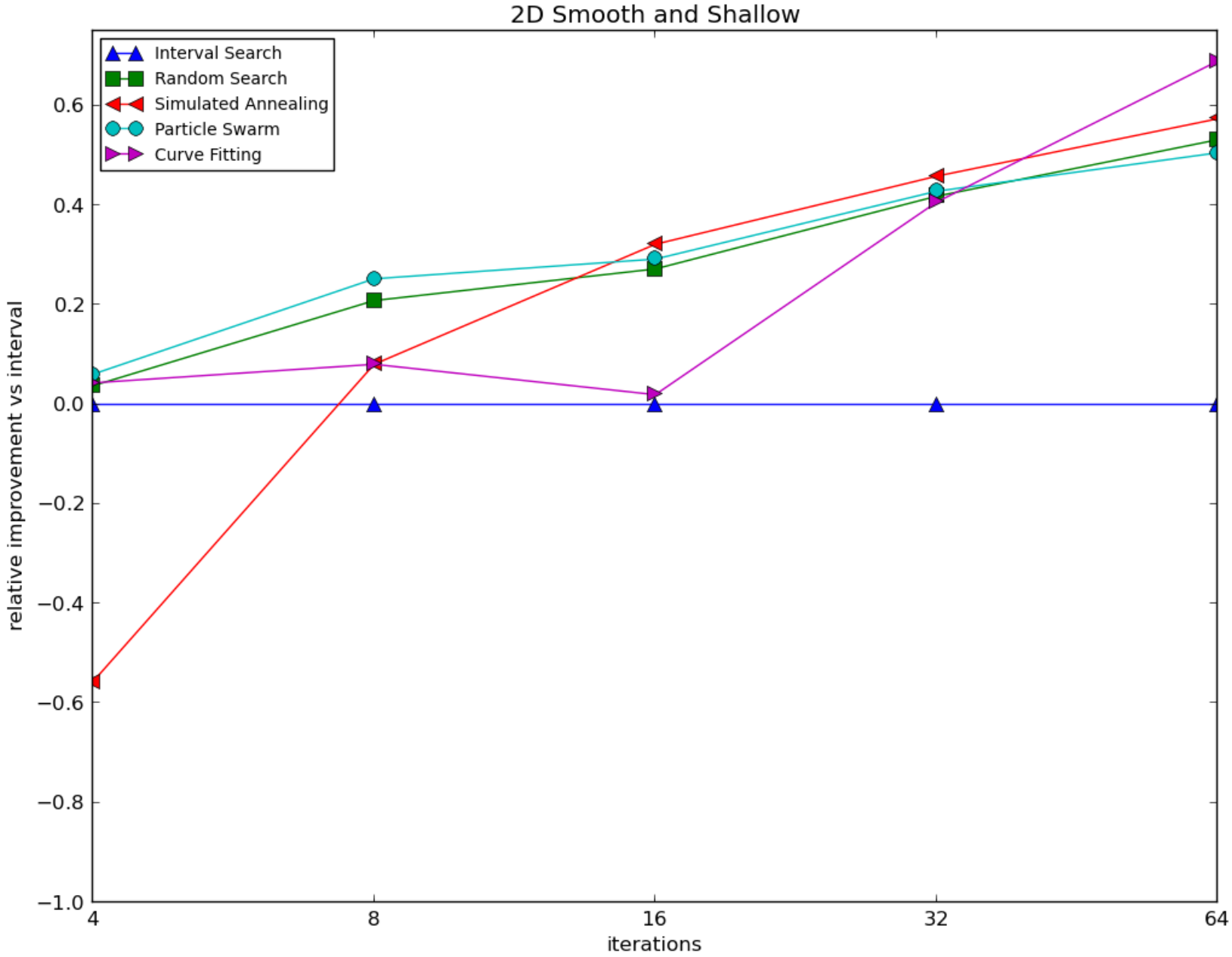

1D Sharp and Steep
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
mean relative solution
1.000
0.100
0.010
0.001
4
8
16
32
64
iterations

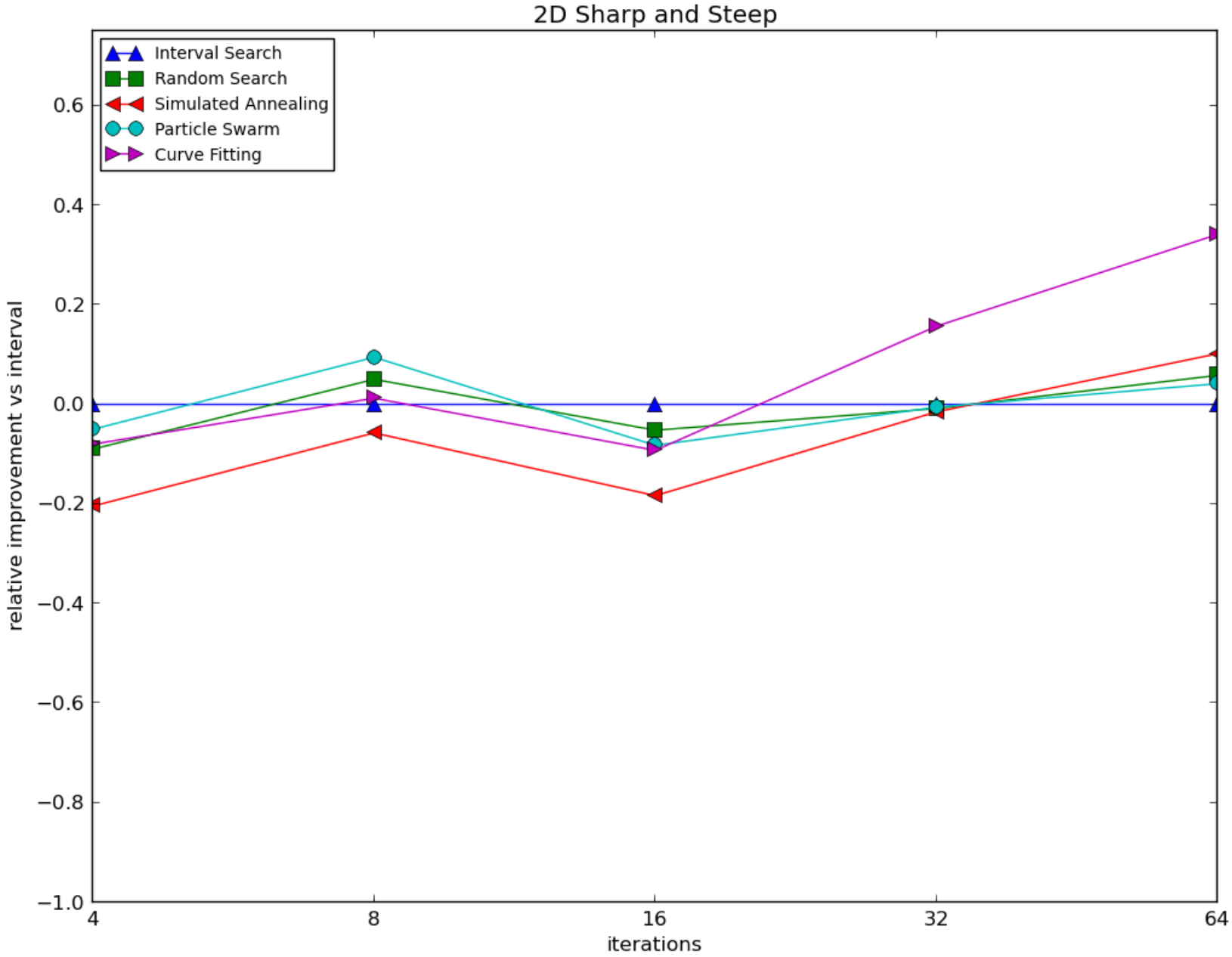

2D Smooth and Shallow
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
mean relative solution
1.000
0.100
0.010
0.001
4
8
16
32
64
iterations


2D Sharp and Steep
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
mean relative solution
1.000
0.100
0.010
0.001
4
8
16
32
64
iterations


1D Smooth and Shallow
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
relative improvement vs interval
iterations
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
4
8
16
32
64


1D Sharp and Steep
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
relative improvement vs interval
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
4
8
16
32
64


2D Smooth and Shallow
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
relative improvement vs interval
iterations
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
4
8
16
32
64


2D Sharp and Steep
Interval Search
Random Search
Simulated Annealing
Particle Swarm
Curve Fitting
relative improvement vs interval
iterations
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
4
8
16
32
64

Conclusions

- Performing more iterations produces a better result
- Smooth and shallow search spaces are optimised in fewer iterations than sharp and steep spaces.
- Optimising in multiple dimensions is less efficient that optimising independently
- There is no single algorithm that produces the best result in all cases
- When performing very small numbers of iterations (<=4) interval search is best
- When performing medium numbers of iterations (5 to 16), random performs slightly better than interval search.
- Particle swarm and simulated annealing do not provide any significant advantage over random search on these problems
- Curve fitting provides the best results when large numbers of iterations are available (>32)

Multi-Method Approach

There is no universally optimal technique for all iterations counts:

- Interval search is generally best for 4 or fewer iterations
- Random search is marginally best between 8 and 16 iterations
- Curve Fitting search is best at the iteration counts of 32 or more.

None of these techniques are mutually exclusive, meaning it is possible to combine them, e.g.

1. First 4 iterations are added selected regular intervals
2. Next 12 iterations are selected at random (avoiding repetitions)
3. All further repetitions are selected by curve fitting using all previous input data

This technique would potentially produce a better average result than using only single technique.

Compiler Flags

Compiler Flags – a different problem?

Compiler flag optimisation is a combinatorial problem, similar to the travelling salesman problems. Very large parameters spaces.

Most applicable techniques are Simulated Annealing and Random Search (as a reference set).

Must always begin with a working set of flags (i.e. with the constraints), these are usually at very low optimisation levels, e.g. O0

To simplify, pre-compile all files with one of a set of N flags, keep the object files. It is then possible to only re-link the application each time with different combinations of flags.

Be careful to around IPA or in-lining for this process as it can distort the results.

Test Datasets

It was not possible to generate a real-world data set by exhaustively searching compile options.

Instead we generated two simulated data set with properties that emulated the effect of compiler flags. Speed improvements + Potential model failures (e.g. numerical blowup, bit-reproducibility problems)

1. The Small dataset – 8 file x 4 Option sets.
   2 File+Option combinations cause “Model Failure” (6%)
2. The Large dataset – 1024 file x 5 Options sets.
   339 File + Options combinations cause “Model Failure” (6%)

Ran each experiment for 1000 iterations to collect average performance figures.

Each simulation starts from worst case scenario (score 1.0)

Final results calculated versus global minimum (score 0.0)

Algorithms – Random Search

Each iterations each file is assigned a random compiler options.

The model is relinked and a new binary generated

The fastest overall result is recorded.

Algorithms Simulated Annealing

The calculation of temperature, *T*, remains the same, depending upon λ, and the maximum T_{max}. The probability of moving between states also remains the same. i.e. the Metropolis Method

$$P(\delta E) = \exp(-\delta E/T)$$

The distance, *d,* becomes the probability that any element will be changed to a random value in any single iteration.

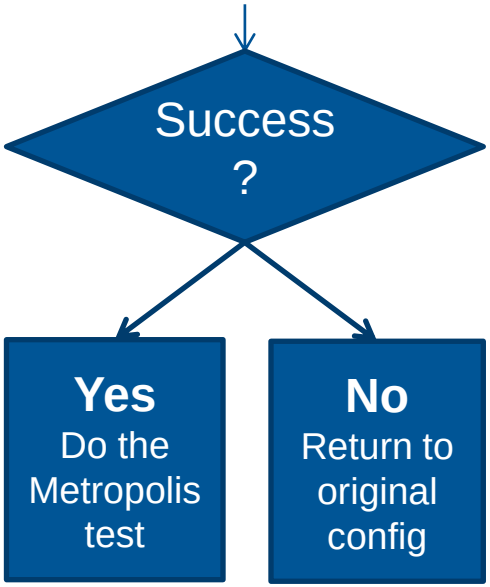
e.g.
d = 0.5 => 50% of files will have their option changed
d = 0.01 => 1% of files will have their option changed

Simulated Annealing - Example Step

<0.5?

Solve_circle.o	O0
Verify.o	O1
readData.o	O0
Purge.o	O0
Curl.o	O2
Divergence.o	O0
Scale.o	O0
Integrate.o	O1
RungeKutta.o	O0
Iteration.o	O0
Main.o	O1

Solve_circle.o	**O1**
Verify.o	O1
readData.o	O0
Purge.o	**O1**
Curl.o	**O2**
Divergence.o	O0
Scale.o	O0
Integrate.o	**O1**
RungeKutta.o	O0
Iteration.o	**O2**
Main.o	**O0**



Results – Mean Relative Solution

Small Dataset

Iters	Random	Sim Anneal d=0.64	Sim Anneal d=0.01
4	0.411	0.431	0.977
8	0.294	0.190	0.962
16	0.210	0.068	0.917

Large Dataset

Iters	Random	Sim Anneal d=0.64	Sim Anneal d=0.01
4	1.0	1.0	0.988
8			0.976
16			0.953
32			0.901
64			0.830
128	1.0	1.0	0.701
256	1.0	1.0	0.518

Analysis – Why is there no progress on the larger case?

For the large dataset, the 6% of file/option combinations are bad.

Therefore the probability of choosing a "good" file/option for any individual files is:

$$P_{individual} = 1.0 - 0.06 = 0.94$$

The probability of selecting 1024 "good" file/option combintions =

$$P_{1024} = (1.00 - 0.06)^{1024} = 7.01 \times 10^{-19} \text{ (Extremely Unlikely)}$$

The probability of selecting a "good" combination when using d=0.64 is:

$$P_{0.64} = (1.00 - 0.06)^{1024 \times 0.64} = 2.41 \times 10^{-12} \text{ (Extremely Unlikely)}$$

Finally the probability when d=0.01

$$P_{0.01} = (1.00 - 0.06)^{1024 \times 0.01} = 0.658 \text{ (Possible)}$$

We can't make progress with big datasets if d is too large!

Conclusions

Search of compiler options randomly will not help if there is even a small proportion of “failure” states

Simulated Annealing is a much more successful technique for approaching this problem.

It might be better to keep P constant, the probability that any iteration will find a successful solution.

This would require estimating the number of “failure” states from the available sample. Would effectively mean dynamically resizing “d” each iteration.

Would benefit from further investigation

Acknowledgements

Thanks to Dr J Beech-Brandt, Dr A Hart and Dr H Richardson of Cray UK in the preparation of this material

This work has been supported by the CRESTA project that has received funding from the European Community's Seventh Framework Programme (ICT-2011.9.13) under Grant Agreement no. 287703.

Any Questions?