

Case Studies in Automatic Design Optimisation using the P-BFGS Algorithm

David Abramson †, Andrew Lewis §, Tom Peachey †

† School of Computer Science and
Software Engineering
Monash University,
Clayton, Vic 3168, Australia

§ High Performance Computing Facility
Griffith University,
Nathan, Qld, 4111, Australia

ABSTRACT

In this paper we consider a number of real world case studies using an automatic design optimisation system called Nimrod/O. The case studies include a photo-chemical pollution model, two different simulations of the strength of a mechanical part and the radio frequency properties of a ceramic bead. In each case the system is asked to minimise an objective function that results from the execution of a time consuming computational model. We compare the performance of an exhaustive search technique with a new non-linear gradient descent algorithm called P-BFGS. The exhaustive search results are produced using enFuzion, a commercial version of the parametric execution software Nimrod. P-BFGS is a parallel variant of the well-known BFGS algorithm and has been tested on a 64 processor Pentium cluster. The results show that P-BFGS can achieve a speedup when compared to the exhaustive search on 3 out of the 4 problems. In addition, it always uses fewer processors than an exhaustive search.

INTRODUCTION

Computational Science and Engineering (CS&E) involves the use of programs that find solutions to mathematical descriptions of real world problems. Where a simulation can be constructed that models the important physical properties of the real system it allows designers to explore a range of scenarios without the need to build a physical prototype. CS&E has been used extensively in the aeronautical and automotive industries, electronic and CAD and environmental modelling. CS&E is often computationally expensive, because, in order to achieve accurate modelling of a problem, it is important to use a high resolution in the mathematical decomposition. Thus, it is not surprising that supercomputers have been used extensively to facilitate CS&E. An important mode of use for CS&E models is in exploring some design space. When used in this way, it is necessary to run the same model many times, each time varying some key parameter. Distributed computers form an excellent platform for this type of work because the model runs concurrently on different data sets. An example of this mode of use is described in [6].

For some years we have been developing a software tool called Nimrod, which allows a user to consider a range of different input scenarios for an arbitrary computational model [1]. Nimrod uses a declarative “plan” file to describe the parameters that are to be varied and the tasks that are required to run the model. Parameters can be ranges or lists of simple types, such as integers, floats and strings. Nimrod takes the cross product of the parameter values and generates a unique set for each simulation run. It then runs the model on distributed computers in parallel. A “Grid Aware” version of Nimrod called Nimrod/G [4] can execute models on a distributed computational grid [2]. A commercial version of Nimrod, called EnFuzion, is targeted at machine clusters [23], and a spreadsheet interface for Nimrod is being developed, called ActiveSheets [7]. Nimrod and EnFuzion have been very successfully applied to a range of case studies in which CS&E models are run against a range of input scenarios, and the outcomes are compared and evaluated [14][6][3][17][4].

However, when there are a large number of different parameter values, the design space can become enormous. To combat this, we have been developing a new tool called Nimrod/O, which searches the space selectively rather than exhaustively with Nimrod [5]. Nimrod/O employs a number of different search heuristics, and one of these, P-BFGS, will be outlined in this paper. Nimrod/O attempts to minimise an objective function value, which is computed either directly by the CS&E model, or as a result of post processing the model output. This means that it can be used to answer design questions such as “which set of parameters will minimise the output of my model?”.

In this paper we will discuss the details of a number of different case studies that we have trialed on Nimrod/O, and will also give details of the P-BFGS search algorithm that we have developed.

CASE STUDIES

Smog

SMOG is a photo-chemical pollution model originally developed by McRae et al. [20]. It has been applied to a number of real world environmental modeling exercises, including some Australian cities [16]. SMOG computes the concentrations of various chemical species in a 3 dimensional airshed. A number of sensitivity experiments are typically performed, in which the levels of pre-cursor species are scaled to determine the effect on the resulting pollution. The results of this work are often portrayed in a 2 dimensional “dodge” plot diagram, as shown in Figure 1, which shows how key pollutants such as Ozone vary as the pre-cursor species are throttled. When used in this case study we posed the following question: “What pre-cursor scaling factors will minimise the production of Ozone in the airshed?” The answer to this question is non-trivial because the chemistry is non-linear, and thus simply reducing the pre-cursor levels does not guarantee a minimum objective function.

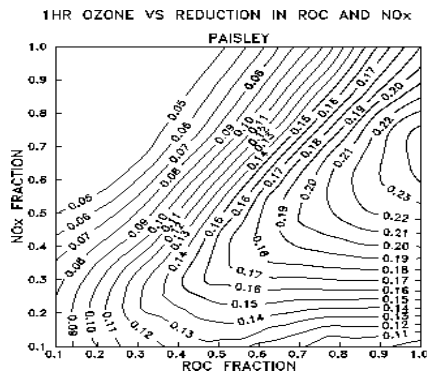


Figure 1 – A Dodge Plot of Ozone against precursors

Plate

“Damage tolerant” design is the design of load-bearing structures that contain flaws before being put into service. Software developed by Chaperon et al. [8] simulates such structures by performing finite element analysis with cracks placed at a range of locations. The PLATE model applies this software to a thin plate under biaxial loading containing a hole of a given width (Figure 2) to determine the hole shape that gives optimum static strength.

Previous work [8] used variable control points to specify the hole profile. The present study used instead a parametric model for the hole. Two variants were tried: S-Plate and GPlate. In the absence of cracks it is well-known an ellipse gives the optimal shape. S-Plate uses the family of ellipses with parameter a , $x = 10 \cos t$, $y = a \sin t$.

t . C-plate allows more complex smooth shapes using three parameters $x = 10 \cos t$, $y = a_1 \sin t + a_2 \sin 3t + a_3 \sin 5t$.

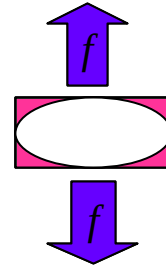


Figure 2 – Biaxially loaded plate with hole.

Ceramic

This case study concerns the design and characterisation of mobile telecommunication handsets. The structure of most mobile telephone antennas includes the chassis as an earth reference. Due to the chassis forming an integral part of the radiating structure, several difficulties are encountered in the measurement of mobile telephone antennas since they are generally connected to a network analyser via a coaxial cable. However, the coaxial cable used to feed a handset has significant effects on both the radiation pattern and resonant frequency [18].

A high permittivity ceramic bead on a straight wire transmission line can be used to suppress propagating cable currents at RF frequencies. This suggests the use of such a bead on the coaxial feed cable to minimise distortion of the radiation pattern during testing. In our antenna testing application, we wish to have minimum transmission through the bead at 1.0GHz. The FDTD method is used to perform a full-wave analysis of a cable structure. The dimensions and properties of the bead were optimised for minimum insertion loss to improve the effectiveness of the bead in minimising cable effects, and the structure is shown in Figure 3.

THE BFGS AND P-BFGS ALGORITHMS

BFGS

The general optimization problem is to minimize a real-valued function f of N variables. In the case of a numerical simulation, the N variables are the input parameters defining the search space to be investigated, and f is some “cost” function of the results of the simulation. In many cases it is necessary to reduce the raw results of the simulation from a large volume of data to a single, real value.

For the problems arising from engineering and scientific modelling, of the type discussed in Section 2, the values computed for f generally form a non-linear, continuous function, for which gradient descent algorithms can be applied. These methods use the derivative of the cost function, as well as its value, to select a search direction, essentially reducing the multivariate optimization problem to uni-variate minimisation along a search vector; a line search. Thus, they consist of two main operations that are executed repeatedly, gradient calculations and line searching. The algorithm chosen for this study is based on the quasi-Newton BFGS method, widely regarded as an efficient and robust gradient descent method for use with continuous functions [12][22][9][21].

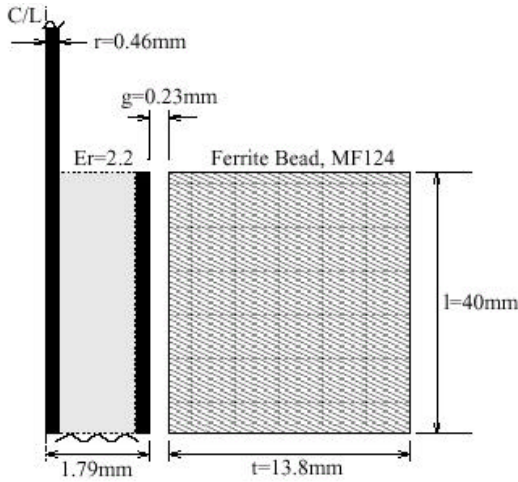


Figure 3 – ceramic bead simulation

The BFGS method maintains an approximation to the inverse of the Hessian, H , of f , where:

$$H_{ij} = \partial^2 f / \partial x_i \partial x_j$$

New approximations to the solution vector, x , are derived by:

1. Compute a search direction, $d = -H^{-1} \nabla f(x)$
2. Find a new x , using a line search, i.e. $x_+ = x + l d$
3. Update the inverse of H using the current approximation to H , x and x_+

For the practical implementation of the method used in the studies reported in this paper, convergence to the solution occurs when the cost function ceases to decrease below a user supplied tolerance, and when there is a sufficient decrease in the Euclidian norm of the gradient of the cost function from its value at the initial starting point. If the latter condition is not met, the method is restarted at least once with the Hessian reset to the unit matrix. The use of

sufficient decrease in gradient norm to confirm convergence, in addition to small step change, is motivated by the idea that if the second derivative of the cost function is well conditioned at the terminating point, a small gradient norm implies a small error norm, and termination is reasonable.

P-BFGS – Parallel Optimization

Because most CS&E simulations are time consuming, we are interested in accelerating the search by parallelising the BFGS algorithm itself. The advantage of this approach is that no modification to the internal code of the simulations themselves is required; they can be treated as black boxes.

When the cost function is the computed result of a numerical simulation, finite difference approximations to the derivative are usually employed. The gradient calculation phase of the algorithm can be parallelised because the individual points in the finite difference stencil can be computed concurrently. The degree of parallelism achievable is determined by the dimensionality of the problem domain.

For unconstrained optimization the line search phase is often inherently sequential, but for the class of problems under consideration there will often be restrictions on the expected range of variables determined by physical constraints on their possible values. The restricted expected range of the variables allows the formulation of these problems as nonlinear optimization problems with simple bounds. The search vector is derived from the gradient and Hessian as in the standard BFGS method. A line segment is then constructed from the current point in the search space, in the direction of the search vector and extending as far as the nearest boundary of the search space. The line search phase of the algorithm may then be parallelised by using a method of interval subdivision, rather than proceeding with the sequential stepping methods typically used.

The parallel form of the algorithm can be summarized by the following stages of initialization, line minimization, convergence check and BFGS update. In pseudo-code these expand to:

Initialization

- Evaluate the objective function at the starting point;
- Calculate the gradient at the starting point by finite difference approximation (parallel function evaluation) and calculate the Euclidian norm of the gradient;
- Set the inverse Hessian to the unit matrix;
- Set the initial line direction to the inverse of the gradient;

Perform Line Minimization

```
While bracket width is greater than desired tolerance do  
    Truncate the line search vector for physical constraints;  
    Determine the nearest boundary in the search direction  
    and set maximum excursion accordingly;  
    Sub-divide the interval as desired (usually an integer  
    multiple of processors available);  
    Evaluate the objective function at the sub-intervals  
    (parallel function evaluation);  
    Select a bracket of three points containing the minimum  
    value;  
end while
```

Test for Convergence

```
if (step change in function value and largest step change  
in position in any dimension are less than the desired  
tolerance)  
then  
    Calculate the norm of the gradient at the "minimum"  
if (norm of gradient subtracted from norm of initial  
gradient is less than an empirically determined constant)  
    then  
        optimization is complete else  
if (this is not the first time convergence test at this point)  
    then  
        optimization is complete else  
        Reset the inverse Hessian to the unit matrix, reset the  
        line search direction to the inverse of the gradient and  
        repeat line minimization
```

Perform BFGS update

```
Calculate a new gradient at the line minimum  
Calculate the step change in the gradient  
Apply the BFGS update to the inverse Hessian  
Calculate the new line search direction  
Repeat Line Minimization and Test for Convergence
```

Inspection of the algorithm outline shows that the degree of parallelism is limited by the desired tolerance of the solution during the line minimization phase, and to the number of dimensions during the gradient evaluation phase. During line minimization, interval sub-division is repeated until the sub-intervals are less than the desired tolerance. So

for a given fractional tolerance, t , the maximum number of CPUs that can usefully be employed is $1/t$. Since most line searches will be less than the entire span of the search domain, to a first approximation the average number of CPUs usefully employed would be $1/(2t)$.

Unlike a sequentially stepping algorithm, the parallel algorithm is less prone to become trapped in some local minima. This is illustrated in Figure 4, which shows the sequence of points given by a sequential algorithm using a combination of stepping and parabolic interpolation, and by the parallel algorithm using interval sub-division. The sequential algorithm terminates in the first minimum found, while the parallel algorithm, extending its search to the boundary, finds the lower minimum.

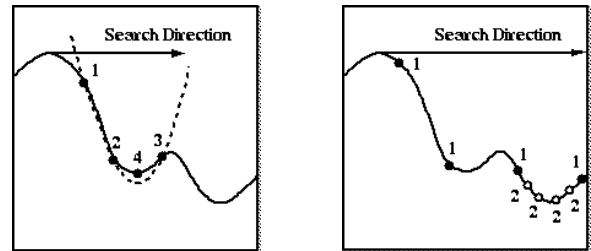


Figure 4: Sequential and parallel line minimization

Given the sources of concurrency in the algorithm, the speedup achieved through parallelisation of the algorithm alone is likely to be modest. However, because the surface is non-linear, it is necessary to perform more than one search in order to try and maximise the chance of finding the global minimum rather than just a local optimum. Accordingly, it is important to run the algorithm more than once, starting it at a different point in the search space each time. Because these searches are independent, they can also be performed in parallel, thus increasing the average speedup. This attribute will become evident in the next section, where we evaluate the performance of the P-BFGS algorithm on the case study problems.

PERFORMANCE RESULTS

Table 1 shows the results of performing an exhaustive search on each of the problems using EnFuzion. Whilst the parameters are continuous, by using a discrete step it is possible to transform the problem into a discrete one. Accordingly, this allows us to perform an exhaustive search across all of the discrete points. The "Func Evals" column shows how many times the program was run to achieve the result shown in the "Optimal Cost" column. The time in seconds is shown for a single execution, and the time for a total run is shown, assuming that it is executed across 64 processors. Whilst the SPlate problem is quite fast, the

other 3 problems take quite significant times to solve. In particular, “Smog” requires 64 processors for nearly 2 months to produce a solution to the required accuracy.

Problem	Num Dims	# Func Evals	Optimal Cost	Time for 1 Func. Eval. (Secs)	Num of CPUs	Wall Clock Time (Hrs:Mins: Secs)
Smog	2	10000	68	31680	64	1375: 0: 0
S-Plate	1	201	15.0	204	64	0: 10: 40
C-Plate	2	2541	14.7	326	64	3: 35: 35
Ceramic	3	12896	-39.8	3000	64	167: 55: 0

Table 1 – Exhaustive Search of Problems

Table 2 shows the results of running the problems using the P-BFGS algorithm. All but the “Smog” problem were run 8 times (concurrently so that the wall clock time was the same as for a single run), but the P-BFGS algorithm performed so well on the “Smog” problem that it was only necessary to run it once.

For each problem, we show the best cost, worst cost and mean. An “*” in the “Opt” column indicates that at least one of the simultaneous executions managed to find the optimal cost, as reported in Table 1. The “# Func Evals for Best” column shows how many times the model was run in the case that produced the optimal result, and the “#Iterations for Best” shows how many times the algorithm recomputed the gradient.

The “Ave # Processors” is computed by taking the average number of processors for an individual run and multiplying it by the number of concurrent starts. All of these are less than 64 in comparison with the exhaustive search that used the entire machine.

The ratio of “Wall Clock Exhaust” to “Wall Clock Best” compares the time taken to compute the solution exhaustively using EnFuzion against using the PBFGS search algorithm in Nimrod/O. Notably, Nimrod/O is able to find the optimal solution to “Smog” and “Ceramic” much faster than EnFuzion, and it does this with fewer processors. In particular, “Smog” only requires 5.1 processors on average to solve the problem 17.4 times faster! GPlate is solved no faster with Nimrod/O than EnFuzion, but it does use fewer processors, and S-Plate is actually slower with Nimrod/O. However, given that the run only takes 34 minutes, the difference is not significant. “Smog” and “Ceramic” seem to perform well because the surfaces are quite smooth, and thus the gradient is a good indicator of the direction of the minimum.

CONCLUSION

In this paper we have attempted to do three things. The first is to introduce the area of using an automatic tool for performing design optimisation. Whilst this technique is not common, we believe it will become more popular as robust, flexible, CS&E models are developed. Second, we have described a set of real world case studies and how they can be addressed using a tool like Nimrod/O. Finally, we have presented a new parallel variant of the BFGS algorithm and showed how it can be applied to the case studies.

The results are encouraging in that they demonstrate that speedup is possible on some problems. However, it is likely that BFGS is not always the best algorithm to apply. For example, if the objective function is not smooth, then the gradient operator will not produce reliable search directions. Accordingly, we have built Nimrod/O in such a way that more optimisation algorithms can be integrated. This will be reported in a future paper.

ACKNOWLEDGEMENTS

The Nimrod/O project has been supported by the Australian Research Council, as part of an ARC Collaborative Grant. We wish to acknowledge our partners in this grant, namely the Environment Protection Authorities of Queensland, Victoria and Western Australia. We wish to thank Turbo Linux Inc. for access to the enFuzion software, and the Distributed Systems Technology CRC, for access to Nimrod/G. We also acknowledge our research partners in the Globus group, in particular Ian Foster from Argonne National Laboratories.

The case studies discussed in this paper involved the work of a number of colleagues. We wish to acknowledge:

- Dr Martin Cope, Dr Steven Spencer and Dr Maciej Skierski for their assistance with the Smog study;
- Professor Rhys Jones, Dr John Sawyer and Mr Paul Chaperon for their assistance with the Plate study;
- Mr Seppo Saario and Dr Jun Lu for their assistance with the Ceramic studies.

REFERENCES

- [1] Abramson D., Sosic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [2] Abramson, D. "Parametric Modelling: Killer Apps for Linux Clusters", Linux Journal, May 2000.
- [3] Abramson, D., Foster, I., Giddy, J., Lewis, A., Sosic, R. Sutherst, R. and White. N., "The Nimrod

Computational Workbench: A Case Study in Desktop Metacomputing”, Australian Computer Science Conference (ACSC 97), Macquarie University, Sydney, Feb 1997, pp 17 - 26.

- [4] Abramson, D., Giddy, J. and Kotler, L. “High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?”, International Parallel and Distributed Processing Symposium (IPDPS), pp 520- 528, Cancun, Mexico, May 2000.
- [5] Abramson, D, Lewis, A. and Peachy, T., “Nimrod/O: A Tool for Automatic Design Optimization”, The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, 11 - 13 December 2000..
- [6] Abramson, D., Power, K., and Sasic, R., “Simulating Computer Networks using Clusters of PCs”, HPC-TelePar'99 at the 1999 Advanced Simulation Technologies Conference (ASTC'99), April 11-15, San Diego, California, USA
- [7] Abramson, D., Roe, P., Kotler L and Mather, D., “ActiveSheets: Super-Computing with Spreadsheets”. 2001 High Performance Computing Symposium (HPC'01), part of the Advanced Simulation Technologies Conference, April 22-26, Seattle, Washington (USA).
- [8] Chaperon, P, Jones, R, Heller, M., Pitt, S. and Rose, F. “A methodology for structural optimisation with damage tolerance constraints”, Journal of Engineering Failure Analysis, 7, pp 281-300, 2000.
- [9] Fletcher. R. “Practical Methods of Optimization.” 2nd ed., Wiley, New York, 1987.
- [10] Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In The Grid: Blueprint for a Future Computing Infrastructure, pages 259-278. Morgan Kaufmann Publishers, 1999.
- [11] Foster, I., Kesselman, C., *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11(2):115-128, 1997.
- [12] Gill, P.E., Murray, W. and Wright. M, “Practical Optimization”. Academic Press, London, 1981.
- [13] Grimshaw, A. and Wulf, W., et al, “The Legion vision of a worldwide virtual computer”, Communication of the ACM, 40(1):39-45, 1997.
- [14] <http://hathor.csse.monash.edu.au/RJK/>
- [15] <http://www.globus.org/>
- [16] Lewis, A, Abramson, D and Simpson, R., “Parallel non-linear optimization : Towards the design of a decision support system for air quality management”, IEEE Supercomputing 97, San Jose, 1997.
- [17] Lewis, A., Abramson D., Sasic R., Giddy J., “Tool-based Parameterisation : An Application Perspective”, Computational Techniques and Applications Conference, Melbourne, July 1995.
- [18] Lewis, A., Saario, S., Abramson, D. and Lu, J., “An Application of Optimisation for Passive RF Component Design”, Conference on Electromagnetic Field Computation, Milwaukee, June 4-7th, 2000.
- [19] Litzkow, M., Livny, M. and Mutka, M.W., “Condor – A Hunter of Idle Workstations”, proceedings of the 8th International Conference of Distributed Computing Systems, pp 104-111, June 1988.
- [20] McRae G.J., Russell A.G. and Harley R.A., 1992. CIT photochemical airshed model -users manuals, Carnegie Mellon University, Pittsburgh, PA and California Institute of Technology, Pasadena, CA.
- [21] Nelder, J.A., and Mead, R., "A simplex method for function minimization", Comput. J., 7, pp. 308-313, 1965.
- [22] Press. W. H., “Numerical Recipes: the Art of Scientific Computing”. Cambridge University Press, 1986.
- [23] Turbo Linux Inc, www.turbolinux.com

Problem	# Starts	Best Cost	Opt	Worst Cost	Mean Cost	# Iterations for Best	# Func Evals for Best	Wall Clock for Best (Hrs:Mins:Secs)	Wall Clock Exhaust Wall Clock Best	Ave # of Processors required
Smog	1	68	*	-	-	2	46	79: 12: 0	17.4	5.1
S-Plate	8	15	*	15	15	2	46	0: 34: 0	0.3	36.8
C-Plate	8	14.7	*	15.7	15.1	7	240	3: 37: 11	1.0	48.0
Ceramic	8	-39.8	*	26.3	-2.1	5	61	15: 0: 0	11.2	27.1

Table 2 – Results of running P-BFGS on various case studies