An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics

David Abramson †, Andrew Lewis §, Tom Peachey †, Clive Fletcher ‡

† Computer Science & Soft. Eng.
Monash University,
CLAYTON, VIC 3800,
Australia

§ HPC Facility
Griffith University,
Nathan, Qld, 4111,
Australia.

‡ CANCES,
University of New South Wales
Australian Technology Park
Eveleigh NSW 1430
Australia

Abstract

In this paper we describe the Nimrod/O design optimization tool, and its application in computational fluid dynamics. Nimrod/O facilitates the use of an arbitrary computational model to drive an automatic optimization process. This means that the user can parameterise an arbitrary problem, and then ask the tool to compute the parameter values that minimize or maximise a design objective function. The paper describes the Nimrod/O system, and then discusses a case study in the evaluation of an aerofoil problem. The problem involves computing the shape and angle of attack of the aerofoil that maximises the lift to drag ratio. The results show that our general approach is extremely flexible and delivers better results than a program that was developed specifically for the problem. Moreover, it only took us a few hours to set up the tool for the new problem and required no software development.

Introduction

Computational science and engineering techniques have allowed a major change in the way that products can be engineered. Rather than building real world prototypes and performing experiments, a user can build a computational model that simulates the physical processes. Using such a model, many design alternatives can be explored computationally in a fraction of the time required. The technique has been applied widely in the areas of aviation, automotive engineering, environmental assessment and electromagnetics.

In the past, we have produced tools that assist a user in performing a rigorous design experiment using an arbitrary computational model. The Nimrod [1] and EnFuzion [2] tools make it possible to describe a number of discrete design scenarios using a simple declarative programming language. The system then produces discrete scenarios, each a unique combination of parameter values from the cross-product of the parameter ranges. If integer or floating point parameters are specified, a step count is used to discretise the domain. The scheme is very powerful, and has been used in real world problems [3]. In order to speed the execution of the experiment, distributed computers are used seamlessly to explore multiple scenarios in parallel. Whilst Nimrod and EnFuzion are optimized for clusters of computers, a "Grid Aware" version of Nimrod, called Nimrod/G [18], utilises resources on a global computational grid [4].

The biggest disadvantage of Nimrod is that when very large search spaces are specified, or when high resolution in the parameter values is required, the number of scenarios may exceed the computational power available. Further, the user may not actually want to explore all design space, but may be satisfied with a "good" solution instead. This background motivated the development of Nimrod/O, a variant of the Nimrod system that performs a guided search of the design space, rather than exploring all combinations. Nimrod/O allows a user to phrase a question like: "What set of design parameters will minimise (or maxmise) the output of my model?". If the model computes metrics like cost, lifetime, etc, then it is possible to perform automatic optimal design.

SC2001 November 2001, Denver (c) 2001 ACM 1-58113-293-X/01/0011 $5.00

Nimrod/O system is almost unique in its ability to solve aribtrary problems without requiring the user to develop optimization code. In general, there are very few systems that allow a user to embed a computational model within a larger problem solving environment, let alone perform automatic optimization. Some systems which have some of these properties are DAKOTA, NEOS, NetSolve and SciRun. The DAKOTA project at Sandia laboratories investigated the combination of optimization with computational models [5]. DAKOTA is a C++ based tool kit that allows a user to choose different optimization algorithms to guide the search for optimal parameter settings. DAKOTA has been successfully demonstrated on structural mechanics applications. However, DAKOTA did not support rapid prototyping and still requires the construction of new programs for each different type of experiment which is performed. NEOS [6] is a distributed, web-based optimization engine that allows a user to solve optimization problems using special remote optimization servers. However, NEOS relies on the objective function being specified in an algebraic form and does not support objective functions implemented by CS&E models. Likewise, NetSolve [7] provides a web based engine for solving linear algebra problems, but does not explicitly address optimization using CS&E objective functions. A number of researchers have investigated the use of Genetic Algorithms for solving complex optimisation problems of the type discussed here, but they do not support multiple search algorithms in the same tool [24]. SciRun [8], and its follow-on Uintah [9], are interactive tools that allow a user to build a CS&E model very rapidly using a graphical programming interface. However, they do not support optimisation. *In contrast, Nimrod/O combines optimization, distributed computing and rapid prototyping in one tool.* The only tool we have found that shares the idea of combining optimisation and rapid prototyping is Optimus, a commercial package produced by Numerical Technologoies. Some details of this are available on the company web site [25].

In this paper we will discuss the Nimrod/O system and its application to a real world case study, the design of a simple, but optimal, aerofoil. We begin with a general discussion of automatic design optimization and the challenges it poses, and then expose the Nimrod/O architecture. We compare the performance of Nimrod/O at solving the aerofoil design against a tailored solution built in Fortran.

Searching for Optimal Designs

Design optimization is not new. There are many examples, particularly from the operations research literature, of the use of optimization theory to find good solutions to real world problems [10]. However, almost all of this work has assumed that the objective function can be expressed algebraically. This means that it is possible to evaluate the function quickly when a new set of design parameters are generated. Further, it is often possible to differentiate the function, which assists algorithms that try to perform gradient descent.

We are concerned with designs that are so complex that their effectiveness can only be evaluated by running a computational model. An important design goal is that the exact nature of the model is not important, and may be solved by a discrete event simulation or the solution of a set of partial differential equations. Because of this generality, it is necessary to run the model from the beginning each time the parameters are changed. Further, if derivatives of the objective function are required, these are typically calculated using a finite difference approximation [11]. Because of this, the cost of executing the optimization algorithm is almost totally dominated by the cost of running the computational model.

From a user's perspective, the problem can be phrased simply - minimise (or maximise) the objective function across a set of parameters. Because almost all real world problems have bounds on the legal parameter values, it is possible to bound the search by these limits. Further, it is often necessary to place further *constraints* on the solution. For example, additional functions that combine parameter values can be used to further reduce the domain. We allow a user to specify both *hard* and *soft* constraints. Hard constraints are enforced during the search process itself. If a hard constraint would be violated by a particular choice of parameter values, then that part of space is not explored. In contrast, soft constraints are implemented by adding a penalty value to the objective function. Accordingly, soft constraints can be violated during the search, but the objective function is artificially higher than if the constraint was satisfied. These techniques are standard in non-linear optimization [12].

In general, the objective functions under consideration are non-linear, may not be smooth and may contain a high degree of noise. In addition, parameters may be continuous or discrete, depending on the nature of the underlying

problem. Thus, no single optimization procedure will work for all problems. Some problems will contain multiple local minima, and it is impossible to guarantee that any one search algorithm will find the global minimum. Accordingly, Nimrod/O supports a range of algorithms, which can be executed multiple times (in parallel) from different starting locations. When a number of algorithms are used, each with different starting locations, it is possible to gain some insight into the nature of the objective function, and to generate a number of potential solution to the problem. Regardless of the search technique that is used, we assume that the computational model is well formulated, stable and robust across the parameter ranges.

Nimrod/O Search Algorithms

At present, we have implemented four optimisation algorithms, namely a gradient search code called P-BFGS [13][12], a Simplex search [20], a Divide-and-Conquer heuristic [13] and Simulated Annealing [23]. In this section we give a brief introduction to each of these algorithms. More details can be found in [13][14]. In the case study reported in the paper we only used two of these algorithms, P-BFGS and Simplex.

P-BFGS

The BFGS optimization method is a quasi-Newton method, that is it is based on the iterative formula

$$x_{k+1} = x_k - \alpha_k H_k g_k . \qquad (1)$$

Here $x_0, x_1, \ldots$ are the successive approximations to the optimal point, considered here as column vectors. g_k is the gradient of the objective function at the point x_k (also a column vector) and H_k is a square matrix described below. The scalar α_k is a positive number chosen so as to minimize the objective $f(x_{k+1})$. The initial value of the matrix, H_0, is the identity matrix, so (1) becomes $x_1 = x_0 - \alpha_0 g_0$ which implies a search directly downhill from x_0. As the search proceeds H_k is updated so that it approaches the "inverse Hessian" matrix. Then (1) contains second derivative information, giving faster convergence than a downhill search, at least if the objective function is sufficiently smooth. The quasi-Newton methods vary as to the updating formula for H_k. For the BFGS method this is

$$H_{k+1} = H_k + \frac{1}{g_k^T c_k}[b_k c_k c_k^T - c_k g_k^T H_k - H_k g_k c_k^T] \quad (2)$$

where $b_k = 1 + (g_k^T H_k g_k)/(g_k^T c_k)$.

We have implemented a parallel variant in which the elements of the finite difference stencil are computed in parallel and the line search is broken into a number of parallel evaluations. The latter modification alters the properties of the algorithm from the original BFGS [12]. BFGS can be applied to both continuous and discrete domains, and works well when the surface is smooth. The concurrency in PBFGS is dependent on the number of parameters and the degree of subdivision of the line search. Typically, we have tested with about 10 processors and have achieved reasonable speedups. When multiple starts are considered, the concurrency is higher.

Simplex

The simplex algorithm implemented in Nimrod/O is a variant of the simplex method of Spendley, Hext and Himsworth [27][20].

The algorithm proceeds as follows. For a search in N dimensional space, the method maintains an N dimensional "simplex", that is a set of $N+1$ points which do not fit into any hyperplane. For example in 2 dimensions, a simplex is a set of 3 points not in a straight line, the vertices of a triangle. In 3 dimensions a simplex is a tetrahedron. Basically the method iteratively replaces the worst vertex by another on the other side of the vertex.

The cost is evaluated at the points of the simplex. Suppose we label the vertices in order of descending cost: x_0, x_1, ... x_m. For convenience we will write *w* for x_0 (the worst), *n* for x_1 (next worst), and *b* for x_m (the best). Let the point *c* be the centroid of all the vertices except *w*. We take *w* and reflect it through *c*, to create new points:

p	an equal distance on the other side of *c*
q	half the distance of *w* from *c*
r	twice as far from *c* as *p*.

These points are all candidates to replace *w* in the simplex, as shown below in Figure 1.

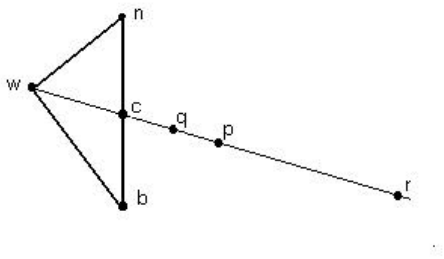

Figure 1 – Simple algorithm extension

We have implemented a parallel variant of the basic algorithm that computes some of the alternative locations for the new vertex in parallel. Whilst this only generates a low level of concurrency, it can accelerate the search process between 2 and 3 times. When combined with multiple starts the algorithm can make effective use of a larger number of processors.

Divide-and-conquer

This method subdivides the domain, evaluates the cost at the subdivision points and uses these values to select a subset of the domain. This sub-domain becomes the new domain for the next iteration of this procedure. The process continues until the variation of cost over the subdivision points is within certain bounds.

The case of a one dimensional search space is simp lest to describe. Let $[a,b]$ be the original domain. We subdivide this interval into M steps using $M+1$ equally spaced points $a = x_0, x_1, ..., x_M = b$ and evaluate the cost at these points. Suppose that the minimum cost occurs at point x_i. Assuming this is not one of the end points, we select x_{i-1} and x_{i+1} as "brackets" for the solution. The minimum sought is assumed to be somewhere between these values. Then a is replaced by x_{i-1} and b by x_{i+1} and the process repeated. In the case where the minimum point x_i is equal to a then the brackets are taken as a and x_1 . In the case where the minimum point x_i is equal to b then the brackets are taken as x_{N-1} and b . The method is known to find the global minimum if the cost function is "unimodular". In most cases however the method should be considered as a heuristic that should find a local minimum.

The "Fibonacci search" method is similar to the method implemented; it uses $M = 3$ steps and points that are *not* equally spaced. Fibonacci search is known to be the most efficient possible in terms of the number of function evaluations. The Fibonacci method is not currently implemented in Nimrod/O.

For the case of an M dimensional search space, the domain for each dimension is subdivided. The number of steps used may vary between dimensions. For the *j*th dimension, suppose that the bounds are a_j , b_j and that M_j steps are used. This gives $M_j + 1$ values for this dimension. Taking all possible combinations of one value from each dimension yeilds a total of $(M_1 + 1)(M_2 + 1)...(M_N + 1)$ points, the "current grid". These are sent for parallel evaluation. The point that gives the minimum cost is the centre of the next iteration. Bracket are taken on either side for each dimension as for the one-dimensional case. These bracket define the subdomain for the next iteration.

A parallel version of the algorithm has been implemented. This variant evaluates the points of the domain in parallel, and thus the concurrency depends on the number of points in a domain. This can be quite high, and when combined with multiple starts, can utilise a large number of processors.

Simulated Annealing

Simulated annealing combines a downhill search with a random search. Suppose that initially we have a point x in the search space and that the cost at that point is f(x). A new point x' is randomly generated that is "nearby" in some sense; we will call this a "trial point". The cost there is f(x'). Next we decide whether to move to x', that is whether to replace x by x' as the current approximation. If f(x') < f(x) then the move is definitely accepted. If f(x') ≥ f(x) then the move is accepted with a probability of

$$\Pr(move\_accepted) = \exp\left(\frac{f(x) - f(x')}{T}\right)$$

Here T is a positive number called the “temperature”. If T is large then this formula gives a value close to 1. If T is small compared to f(x)-f(x'), then this number is very small. In summary, downhill moves are always accepted,

uphill moves may be accepted depending on the size of the change in cost relative to the temperature.

Once the move has been accepted or rejected, the process is repeated with a new trial point. This gives a sequence of current approximations, the "Markov chain" of points. Initially the temperature is set high so that almost all moves are accepted. Gradually the temperature is reduced; when it is low only downhill moves are accepted and the current approximation settles into a local minimum. Simulated annealing is recommended [26] over a pure downhill search when the landscape is rough as the uphill moves at high temperature provide a chance of escaping from a local minimum.

The Nimrod/O parallel implementation provides concurrency in three ways. First multiple starts are run in parallel. Secondly, all evaluations are cached avoiding repetition of the same evaluation. This is particularly important for simulated annealing as trial points may be revisited many times. Thirdly, the implementation uses and experimental parallelisation technique based on "speculative computing". This technique searches a tree of possible moves and evaluates the potential points in a batch.

Nimrod/O Architecture

Figure 2 shows a schematic of the Nimrod/O architecture. Nimod/O accepts a superset of the declarative “plan” files that are used to drive Nimrod, as discussed in [13]. In Nimrod/O additional statements are included that describe the optimization process that is to be used. Figure 3 shows and example of such a plan file, highlighting some of the new statements in Nimrod/O. The `parameter` and `task` statements define the optimisation variables, and the commands required for running the application. The `method` statements define various parameters for the search methods, such as how many concurrent starts are required, the tolerance, etc.

Nimrod/O has been built to support an extensible range of optimization procedures. Each of these procedures requires the evaluation of an objective function in order to proceed. This is performed by a request to the Nimrod/G or EnFuzion remote job execution engines. The algorithm forms a set of parameter values and passes these to Nimrod/G or EnFuzion for evaluation against the model. The model is run on an appropriate platform and the objective function value is extracted from the model output. A cache is superimposed between Nimrod/O and the backend to reduce the number of calculations required if the same parameter values are requested more then once. A persistent database is attached to the cache to support restart if Nimrod/O is terminated prematurely. By storing all function values, the user can restart the system from scratch and proceed to the same position without rerunning the computational models, providing a recovery process in the event of machine or network failure.

Nimrod/G and EnFuzion share a common API, and thus it is possible to execute the Nimrod/O model computations either on a local cluster, or on the Grid, depending on the available resources. This choice is transparent to the algorithms in Nimrod/O, and the selection of backend can be left to the user depending on the available resources and

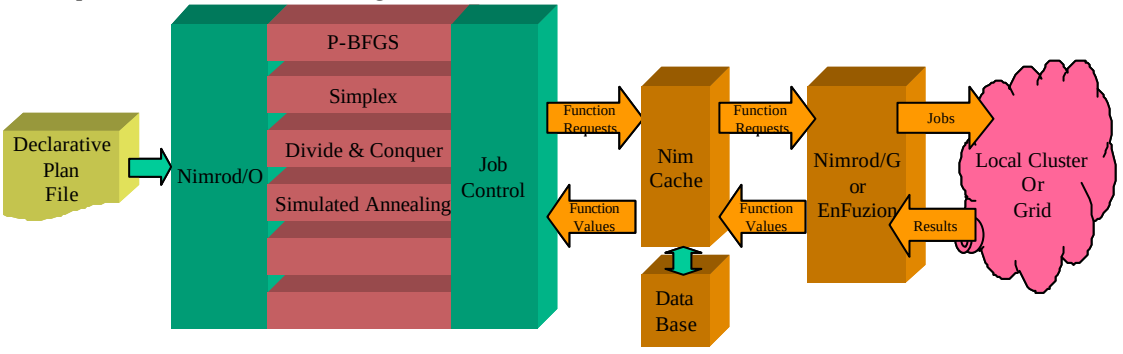

Figure 2 - Architecture of Nimrod/O

the number of processors required. Accordingly, scheduling of these jobs is also left to the backend technology. Figure 4 shows the Nimrod/G and EnFuzion dispatchers in use. In the Nimrod screen on the left, the coloured boxes represent the different instances of the model, and these can be seen assigned to hosts on the Grid. In this particular example Nimrod is supporting three different Grid middleware services, namely, Legion[15], Globus [16] and a Condor [17]. The EnFuzion screen dump shows the different model instances as they run on the various nodes of the local cluster. The differences between Nimrod/G and EnFuzion are discussed in other papers [18].

```
parameter x float range from -20 to 20
parameter y float range from 0 to 40
parameter z text select anyof "Red"
"White" "Blue"

task main
   copy  root:~/projs/* node:.
   node:execute ./run.script $x $y $z >
        final.objfn
   copy  node:final.objfn output.$jobname
endtask

method simplex
   starts 5
      starting points widespaced 3 5
      tolerance 0.10
   endstarts
endmethod

method bfgs
   starts 5
      starting points random
      tolerance 0.10
   endstarts
endmethod
```

Nimrod commands

Nimrod/O specific commands

Figure 3 - A sample Nimrod/O plan file.

An Aerofoil Case Study

Previously, we have applied Nimrod/O to a number of different problems, ranging from air pollution studies, computational electromagnetics and stress analysis [13]. The case study reported in this paper concerns the design of a simple aerofoil. Initially, the problem was developed and solved *without* Nimrod/O at the Centre for Advanced Numerical Computation in Engineering & Science (CANCES). Specifically, a simple two-dimensional aerofoil was modelled using a FLUENT simulation [19]. The shape of the simulation mesh was generated from the problem input parameters, and FLUENT was used to compute the flowfield around the aerofoil, from which lift and drag properties were derived. In addition, a special Fortran program was written to take the solutions from the FLUENT simulation and iteratively search for an optimal wing design using a Simplex method [20]. The aerofoil mesh generated by GAMBIT, had 28089 nodes and 49426 elements, made up of 43090 triangular elements and 6336 quad elements. The convergence criteria for the FLUENT run were set at 1.0e-4. The original purpose of the experiments was to investigate the applicability of optimization to the design of an aerofoil, with the goal of maximising the ratio of lift to drag. The solution took some time to develop because it required the development of code specifically tailored to the problem.

The aerofoil shape and configuration is governed by three parameters - the angle of attack, the camber and the thickness, as shown in Figure 5. At a particular configuration of these parameters, the objective function value, the ratio of lift to drag, is maximised. Complete enumeration of the space is infeasible because the number of simulations required is excessive.

Following the earlier work performed at CANCES, we applied the Nimrod/O tool to the same problem. Again, the original FLUENT code was used to model the aerofoil, however, rather than using the Fortran code developed specifically for the problem, Nimrod/O was used to perform the search. The study was an outstanding success in three ways:

- It only took about an hour to set up Nimrod/O on the Aerofoil problem.
- The results that were generated by Nimrod/O were better than those of the specific code.
- We were able to explore two algorithms by changing only the plan file

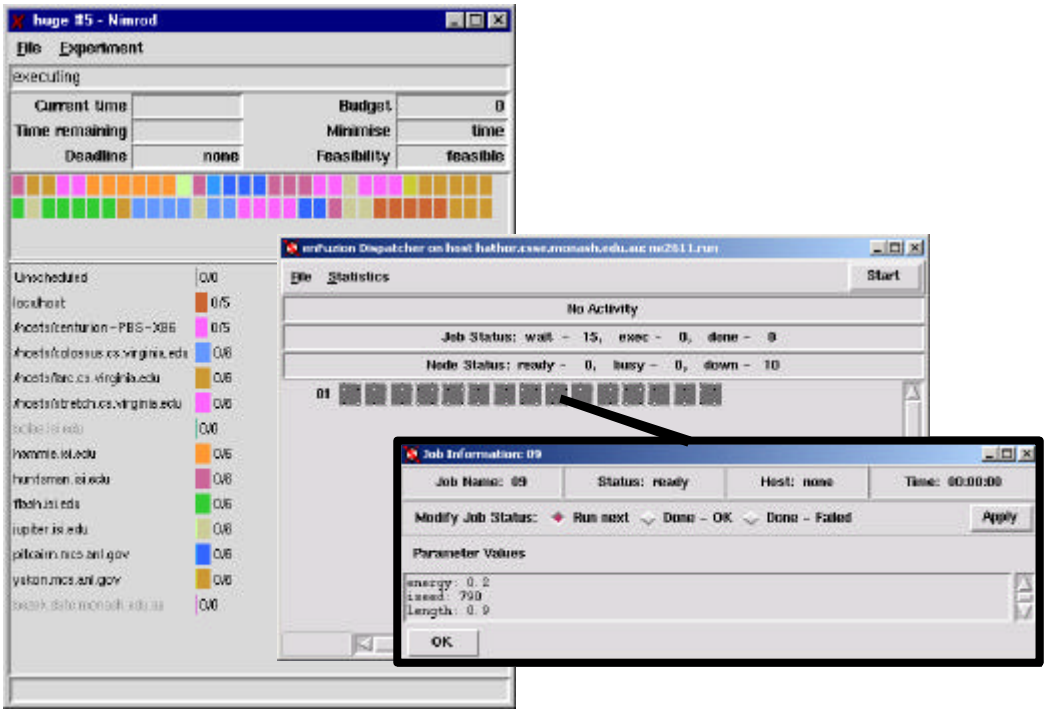

Figure 4 – Nimrod and EnFuzion engines

Table 1 summarises the results of the work. The "Best Objective Function" column shows the highest value of the lift/drag ratio achieved by the algorithm. The "Number of function evaluations" is the number of times the FLUENT simulation was executed. These simulations are much more expensive than the optimisation process itself. Therefore, the bulk of the run time is spent in executing the CFD code rather than performing the optimisation calculations. The table shows that our implementation of Simplex and P-BFGS both returned a better objective function values than the tailored Fortran code, but they required more function evaluations. Further, Simplex performed better than P-BFGS, and returned a value of 71.9 with fewer evaluations. The "Wall Clock Time" shows how long the Nimrod/O computations took on the VPAC AlphaServer SC[1], consisting of Alpha EV68's running at 833 MHz. The Tailored Fortran code ran on a different system, so the "Wall Clock Time" is unknown for this method.

[1] 32 Compaq ES40's (4 processors each) connected by a Quadrics interconnect.

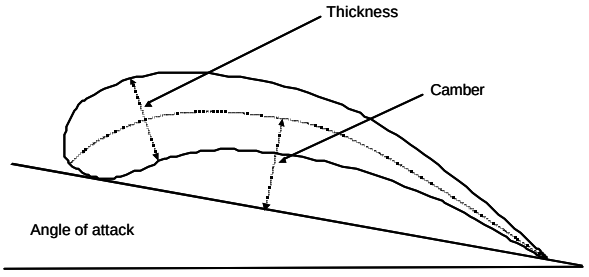

Figure 5 - Aerofoil structure

Table 2 shows the behaviour of the two Nimrod/O algorithms across multiple concurrent searches. Here we see the best, worst and average objective function values, and the number of function evaluations required to achieve them. Interestingly, the average result from the Nimrod/O Simplex is better than the tailored code, and achieved this with fewer function evaluations on average. However, the best P-BFGS still performed better than the average Simplex, but used more function evaluations.

Whilst the P-BFGS algorithm did not perform as well as Simplex, on average, on this particular problem, it is not possible to know this beforehand. *One of the key advantages of Nimrod/O over the tailored solution is that it is possible to try a number of different algorithms on the one problem.*

Figure 6 shows the tracks followed by the Simplex (the black curve) and the P-BFGS (the blue curve) algorithms in the three dimensional space. We have drawn three different iso-surfaces of the objective function, coloured yellow, green and red. The red iso-surface corresponds to an objective function value of 60, green corresponds to 52 and the yellow one corresponds to 47. Thus, in both cases, the tracks show the algorithms iteratively improving the solution until they terminate near the global optimum. Whilst both searches began at similar locations, they followed different paths and converged to different local minima, with Simplex yielding a better result. An interesting observation is that both of the Nimrod/O algorithms that were used yielded good solutions without requiring multiple starting locations. This is presumably due to the relatively smooth nature of the objective function. A movie form of Figure 4 is available on the web at [22]. Figure 7 shows the search paths taken by the 8 Simplex searches each from a different starting location. Whilst they all start at different locations, those that find good solutions converge on the same region of space, surrounded by the red iso-surface. A few of the searches terminate early, some distance from the global minimum.

Figure 8 shows the parallel behaviour of the system running 8 Simplex and 8 P-BFGS searches concurrently. Each of the search methods was run independently on a Beowulf cluster using EnFuzion as the backend technology. Since the machine size was limited to 64 processors, each of the different methods was run independently (as 2 lots of 8 concurrent starts) and then these two runs were added together and graphed. Whilst the current experiment used a local cluster, we have run similar experiments on the Grid using the Nimrod/G engine. Although the concurrency within the search algorithms is relative low, running multiple searches in parallel improves the resource utilization and also solves a given number of runs in less time. Further, we continue our search for efficient search algorithms that have more concurrency.

Conclusion

In this paper we have described a new design tool called Nimrod/O, and have demonstrated its effectiveness on solving a simple, but realistic, CFD problem. In comparison with an existing tailored simplex search, Nimrod/O allowed us to explore two different optimisation algorithms very quickly, and without any code modification. Both of our algorithms out-performed the tailored code, yielding better results in less time. Some low level concurrency in the algorithms further accelerated their performance. The Nimrod/O design philosophy allowed us to perform automatic design without tuning or modifying the existing FLUENT simulation. *The most dramatic result of the work was the ease of application – it only took us about an hour to set up Nimrod/O for a new optimisation problem.*

Method	Algorithm	Best Objective Function	Angle of Attack	Thickness	Camber	Number Function Evaluations	Wall Clock Time (hh:mm)
Tailored Fortran	Simplex	52.8	0.8	0.03	0.10	67	Unknown
Nimrod/O	Simplex	71.9	1.00	0.02	0.10	82	29:21
Nimrod/O	P-BFGS	69.0	1.11	0.03	0.11	113	43:40

Table 1 - Case study results

Method	Algorithm	Best		Worst		Average	
		Objective Function	Number Function Evaluations	Objective Function	Number Function Evaluations	Objective Function	Number Function Evaluations
Nimrod/O	Simplex	71.9	82	49.0	25	65.8	49.0
Nimrod/O	P-BFGS	69.0	113	44.8	99	54.6	76.1

Table 2 – Variance in algorithm behaviour

Clearly our work is in its early stages – the case study shown here is only a very simple design. In a real world engineering environment, one would expect the computational models to be much more complex, involving multiple simulation techniques. Also, an industrial strength problem would have more parameters. How well our work scales can only be determined by experimentation. However, the early results reported here and in [13] are extremely encouraging.

The choice of back end technology allows us to abstract the exact nature of the computational platforms, and thus the user is free to utilise machines ranging from a single high-end workstation through to the computational Grid. For small studies a local cluster provides sufficient performance. However, when multiple searches are required, using different algorithms, the number of concurrent executions exceeds the resources available on a local cluster, and the Grid becomes a viable platform. We expect to perform some larger design experiments in the area of mechanical durability in the near future.

Acknowledgments

This work has been supported by the Australian Research Council (ARC), the Victorian Partnership for Advanced Computing (VPAC) and the Centre for Advanced Numerical Computation in Engineering & Science (CANCES). The programs were run on facilities at CANCES and VPAC. We would like to acknowledge Ivan Mayer for his assistance, and Jimmy Li for his work on the original optimization code.

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”, The Linux Journal, #73, May 2000, pp 84 – 91

[3] Lewis, A., Abramson D., Sosic R., Giddy J., “Tool-based Parameterisation : An Application Perspective”, Computational Techniques and Applications Conference, Melbourne, July 1995.

[4] Foster, I. and Kesselman, C. Globus: “The Grid: Blueprint for a Future Computing Infrastructure”. Morgan Kaufmann Publishers, 1999.

[5] Eldred, M., Outka, D., Fulcher, C. and Bohnhoff, W. “Optimization of complex mechanics simulations with object-oriented software design.”, Proceedings of the 36th IAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, pp 2406 - 2415, New Orleans, LA, April 1995

[6] Czyzyk, J, Owen, J. and Wright, S. “Optimization on the Internet”, OR/MS Today, October 1997.

[7] Casanova, H. and Dongarra, J. “NetSolve: A Network Server for Solving Computational Science Problems”, The International Journal of Supercomputing

Applications and High Performance Computing, Vol 11, Number 3, pp 212-223, 1997.

[8] Parker, S. and Johnson, C. "SCIRun: A Scientific Programming Environment for Computational Steering", Proceedings of IEEE Supercomputing 1995, San Diego, December 95.

[9] McCorquodale, J., Parker, S., Davison de St. Germain, J. and Johnson, C., "The Utah Parallelism Infrastructure: A Performance Evaluation", 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, April 22-26, 2001, Seattle, Washington (USA).

[10] Taha, H., (1992) "Operations Research: An introduction" Macmillan Publishing Company, New York, 5th Ed, 822 pages.

[11] Press. W. H., "Numerical Recipes: the Art of Scientific Computing". Cambridge University Press, 1986.

[12] Gill, P.E., Murray, W. and Wright. M, "Practical Optimization". Academic Press, London, 1981.

[13] 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.

[14] Abramson, D, Lewis, A. and Peachy, T. "Case Studies in Automatic Design Optimisation using the P-BFGS Algorithm", 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, April 22-26, 2001, pp 104 – 109, Seattle, Washington (USA).

[15] Grimshaw, A. and Wulf, W., et al, "The Legion vision of a woroldwide virtual computer", Communication of the ACM, 40(1):39-45, 1997.

[16] Foster, I., Kesselman, C., *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11(2):115-128, 1997.

[17] 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.

[18] 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.

[19] http://www.fluent.com/

[20] Nelder, J.A., and Mead, R.,"A simplex method for function minimization", Comput. J., 7, pp. 308-313, 1965.

[21] Fletcher. R. "Practical Methods of Optimization." 2nd ed., Wiley, New York, 1987.

[22] http://www.csse.monash.edu.au/~davida/nimrodo/aerofoil.mpg

[23] van Laarhoven, P.,Aarts E., (1987): "Simulated Annealing: Theory and Applications" D Reidel Publishing Company, Dordecht, 186 pages.

[24] Poloni, G and Pediroda, V. "GA Coupled with Computationally Expensive Simulations: Tools to Improve Efficiency", Chapter 13, in "Genetic Algortihms and Evolution Strategy in Engineering and Computer Science", Edited by Quagliarella, Periaux, Poloni and Winter, John Wiley & Sons, 1998, ISBN 0 471 97710 1.

[25] http://www.lmsintl.com/

[26] van Laarhoven P.J.M. and E.H.L. Aarts, Simulated Annealing: Theory and Applications, Reidel, 1987.

[27] Spendley, W, Hext, G. R. and Himsworth, F. R. "Sequential applications of simplex designs in optimization and evolutionary operation", Technometrics, 4, pp 441-461, 1962.

Surface	Objective Function Value
Red	60
Green	52
Yellow	47

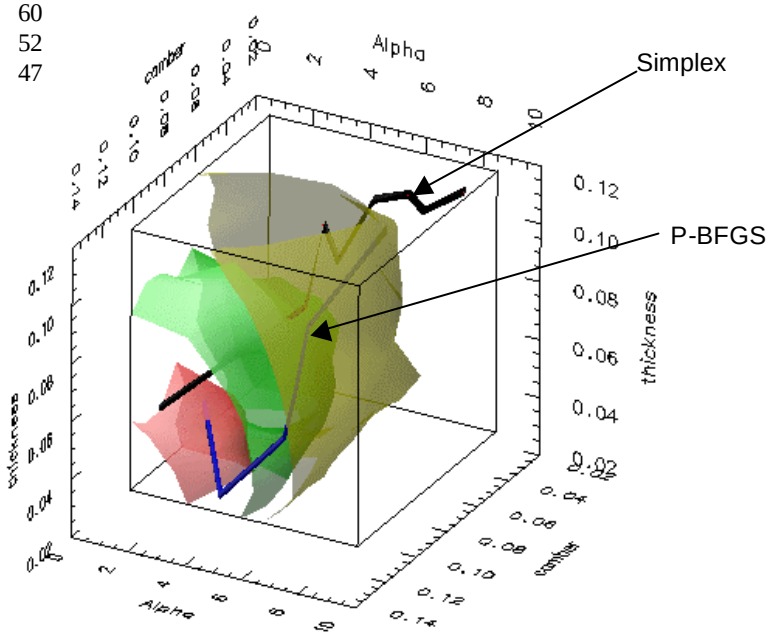

Figure 6– Simplex and P-BFGS search tracks

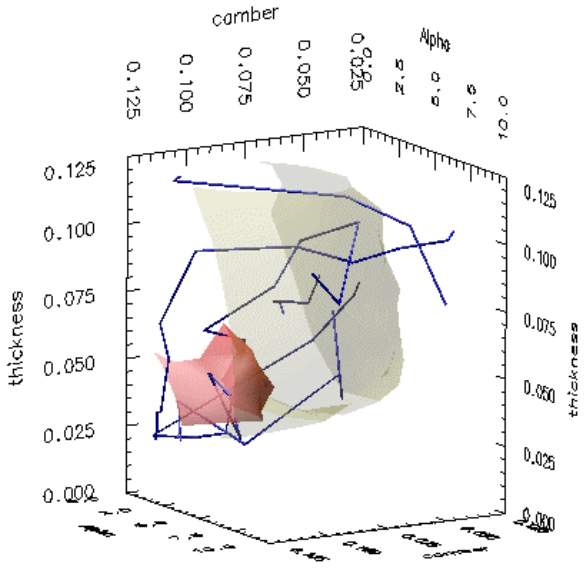

Figure 7 – 8 Simplex search tracks

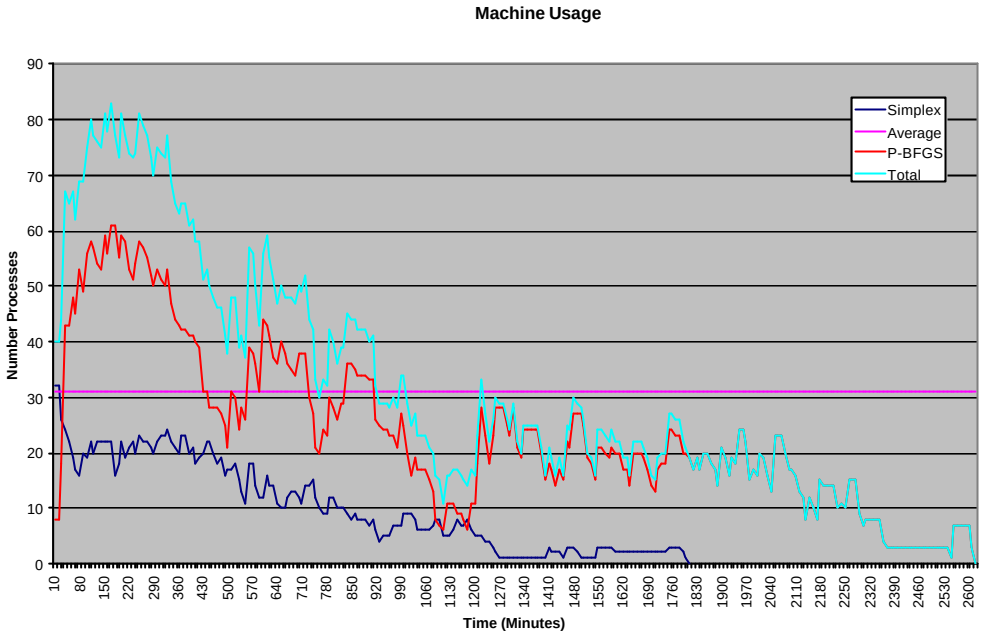

Figure 8 – Parallel Behaviour of multiple searches.