Scientific Programming 14 (2006) 1–11
IOS Press

RSCS: A parallel simplex algorithm for the Nimrod/O optimization toolset

Andrew Lewis[a], David Abramson[b] and Tom Peachey[b]
[a]*Division of Information Services, Griffith University, Brisbane, Qld, Australia*
E-mail: a.lewis@griffith.edu.au
[b]*Department of Computer Science, and Software Engineering, Monash University, Melbourne, Vic., Australia*
E-mail: {davida,tcp}@csse.monash.edu.au

Abstract. This paper describes a method of parallelisation of the popular Nelder-Mead simplex optimization algorithms that can lead to enhanced performance on parallel and distributed computing resources. A reducing set of simplex vertices are used to derive search directions generally closely aligned with the local gradient. When tested on a range of problems drawn from real-world applications in science and engineering, this reducing set concurrent simplex (RSCS) variant of the Nelder-Mead algorithm compared favourably with the original algorithm, and also with the inherently parallel multidirectional search algorithm (MDS). All algorithms were implemented and tested in a general-purpose, grid-enabled optimization toolset.

Keywords: Parallel programming, optimization, Nelder-Mead algorithm

1. Introduction

In scientific and engineering research and design increasingly sophisticated, rigorous and realistic numerical simulations of physical systems are used to understand these systems, and aid in the design process. Engineers can use computational models instead of building physical prototypes to examine the behaviour of components or systems. Such an approach is usually faster and cheaper and hence allows the user to explore various design scenarios. In particular the user may search through combinations of design parameters in order to achieve a design that is optimal in some sense. Similarly, scientific research is increasingly using computational models, and there is often a need to determine those model parameters that produce the best fit to real-world data.

While use of computational models is becoming routine across a wider range of applications, they are often used in an informal manner: an engineer might use a simulation to test a handful of different cases and pick the best of them. More rigorous use to comprehensively explore the design parameter space has the potential to deliver better outcomes. This drives a demand for the capability to perform automatic optimization, minimising or maximising some derived quantity, a measure of "fitness" of the design, to reach a desired objective.

The computation of these objective function values is generally an extremely computationally intensive process when models, which may each take hours to compute, must be run tens, or maybe hundreds, of times to effectively refine possible designs. An ability to deploy the parallel and distributed computing resources that form the basis of contemporary high performance computing architectures would be a distinct advantage in making automatic optimization a practical tool in the engineering design process.

The design engineer who uses the optimization program, while an expert in the application domain, cannot always also be expected to be an expert in computer science. Ideally, to be useful a general purpose optimization tool should be easily applicable to a wide range of problems without assuming specialised knowledge in methods of optimization from the user. The more it can be treated as a "black box", the wider its potential adoption and the greater its end benefit. To meet these needs, the algorithms described in this word have

ISSN 1058-9244/06/$17.00 © 2006 – IOS Press and the authors. All rights reserved

been implemented as components of a fully integrated optimization toolset, Nimrod/O [1–3].

Many different optimization algorithms have been developed, from traditional gradient descent methods [4,5] to more recent innovations inspired by systems in nature, evolutionary and genetic algorithms [6]. Of enduring popularity, particularly for problems with "noisy" objective functions, or where gradient information is unreliable, unavailable or difficult to obtain, are direct search methods. Foremost among these is the simplex algorithm of Nelder and Mead [7].

In the Nelder-Mead simplex algorithm, the $n + 1$ vertices of a simplex of approximations to an optimal point in n-dimensional parameter space are sampled, ordered by objective function value, and an attempt made to replace the worst vertex by reflection through the convex hull of the remaining vertices, using limited sampling along the search direction so defined. Use of the Nelder-Mead simplex algorithm remains current, largely because, on a range of practical engineering problems, it is capable of returning a very good result [8]. It is also robust to small perturbations or inaccuracies in objective function values [9].

Since the original algorithm treats a single vertex at a time, the overall optimization process can be very time-consuming. With the wide availability of parallel and distributed computing resources, an obvious approach to attempt to reduce the total optimization time is to simultaneously consider and relocate several vertices. This paper proposes a method for concurrent execution of a simplex optimization algorithm, and presents results from numerical tests of the revised algorithm on a number of case studies derived from real-world problems in scientific and engineering applications.

2. The Nimrod/O toolset

Nimrod/O is a development of the Nimrod research project [10–12], incorporating automatic optimization into the framework of what was originally a parameter sweep toolset. Nimrod allows a scientist or engineer to succinctly describe their numerical simulation, define parameter ranges and perform automatic explorations of parameter space using enumeration of the cross-product of the defined parameters on parallel or distributed computers.

While an extremely useful tool, Nimrod suffers from the shortcoming of combinatorial explosion of required model evaluations as problem dimensionality and desired solution resolution increases. Nimrod/O avoids this problem by extending the toolset to include automatic optimization in the same, easily usable framework. Nimrod/O is equipped to run a number of very different optimisation algorithms, those described in this paper being just one set. Importantly, different problems are better suited to different algorithms, and Nimrod/O allows the user to try different algorithms (in parallel if the resources are available) and to choose the one that performs best. In this paper we are describing some enhancements to one of these algorithms.

The structure of Nimrod/O and outline of its operation is shown in Fig. 1. It is designed to be modular, the incorporation of additional optimization algorithms being a relatively simple operation. Since the C source code for Nimrod/O is freely available [13], the developer of a new algorithm can rapidly obtain the service of distributed evaluations Nimrod/O provides. Nimrod/O will also provide an extensive constraint parser, cacheing of completed jobs, granularity control for the parameters and statistical summaries of the evaluations.

The simplest way to incorporate an algorithm is to use the "hooks" provided. For example, if the algorithm is coded in the function `userSuppliedOptimization1` (for which a skeleton is provided) then that algorithm may be invoked by the line

```
method special 1
```

in a schedule file.

The user-supplied function will initially be passed a starting point in search space. What it does with this point is defined by the optimization method. For example, in simplicial methods the starting simplex is constructed around the starting point. However, it should be borne in mind that differing starting points are the only differentiation between the multiple, simultaneous optimizations available through Nimrod/O's multi-start feature.

The dimensionality of parameter space is available through a global variable, and its bounds through passed variable attribute structures. The code needs to use an array of structures of type `Point`. Each structure will hold the coordinates of a point in the search space; simultaneous evaluation of the objective function at a number of points constitutes a "batch of jobs". Execution of a batch is performed by passing the array to the function `evalBatchOfJobs`. On return each structure gives the status of the corresponding job, the result of any imposed constraints, the location of the "optimum" found and the value of the objective function at that point. More detail is supplied in Chapter 10 of the Nimrod/O User's Guide [14].

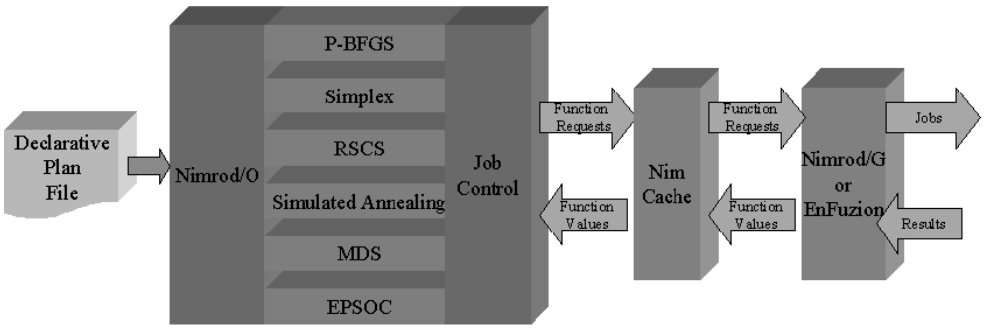


Fig. 1. The Nimrod/O Architecture.

A limitation of this approach is that only the tolerance setting, a value that can be used to decide convergence, is available to the algorithm. Other static, operational parameters of the optimization method may be read from a file, but if the developer wants settings to be passed from the schedule file to the algorithm, it is necessary to modify other parts of the Nimrod/O code. In this case it is advisable to fully incorporate the new method using an existing method as a guide.

Several changes are required. The files involved are shown in parentheses.

- Additions to the parser lexicon to recognise new statements in the schedule (parsesched.c and definitions.h).
- Additions to the structure `Opt_Settings` to include the information required (definitions.h).
- Insertion of parsing options in `ParseInnerLine` (parsesched.c).
- Additions of the new method as a case in `Optimize` (penalties.c).
- Creation of the new optimization algorithm to be called from `Optimize`.

A wide variety of methods is already available in the source code and can be used to provide examples of the code and information needed.

Control of optimization experiments is via a declarative plan file, a simple description of the execution of the numerical model and optimization problem, including parameters, their ranges, and details of the optimization methods to be used. Nimrod/O interprets the plan file, sets up the run environment and passes control to the requested optimization algorithm(s). Nimrod/O explicitly supports simultaneous execution of multiple optimization runs from different starting points, and simultaneous use of multiple optimization methods.

An optimization algorithm, as it runs, passes requests to Nimrod/O job control for objective function evaluations to be performed. These can be grouped in batches where the algorithm is capable of generating multiple concurrent tasks. Job control checks the requested set of parameters against a cache of previously computed results and then automatically dispatches evaluation jobs to computing resources provided by the user, whether they be parallel or distributed computers, via one of a number of job distribution mechanisms. Currently three are provided in the standard toolset:

- Nimrod/G, which is a Globus-enabled tool for use of computational grid resources [15]. Not only does the Grid offer additional computing resources, but potentially also access to specialised software packages and licences unavailable locally.
- an API to EnFuzion, a commercially available implementation of the original Nimrod tool that allows use of parallel computers or collections of workstations on a LAN.
- a mode in which processes are run locally using `fork`(). This mode suits symmetric multiprocessors.

These tools take care of job submission, execution tracking and input and output file handling.

Following is a simple, example declarative plan file. Further details of the syntax can be found in the Nimrod/O Users' Guide.

```
parameter b float range from 5 to 35
parameter tness float range from 2 to 9
parameter bias float range from -0.02 to 0.02

task main
   copy  runfiles/* node:.
   node:execute ./run.script $b $tness $bias
   copy node:result.dat  output.$jobname
endtask

method simplex
   starts 8 named "simplex"
```

```
      starting points random
      tolerance 0.005
  endstarts
endmethod
```

The plan file starts by defining the parameters. A parameter is named, its type defined, how it will be expressed and, where appropriate, the bounds on its value. Supported parameter types are float, integer and text. Float and integer types must be expressed as a range of values, text as a list of values. Where text parameters are used, the listed cases will be evaluated by simple enumeration, i.e. multiple optimizations will be performed, one for each value of the text parameter.

Then, in the section labelled as "task main", a brief description is given of how to run the numerical simulation. Input and output of the simulation are assumed to be via named files. This has been a common method of interaction with large-scale simulations, and is an easy method independent of the model implementation. For each objective function evaluation task input files are copied to the "node" on which execution is to be scheduled. All other files necessary for execution of the simulation should also be copied, including all necessary scripts and executables.

Then a user-provided script is run on the node. It is assumed that the script will provide the commands to run the simulation and derive the single, floating-point objective function value, possibly by post-processing of the model output. It may be noted that the parameters are provided to the script via named environment variables. These will be substituted by the particular parameter values supplied by the optimization algorithm at run-time. The results, written to a file, are copied back to the scheduling node, and placed in a file with the standard name "output" and an extension identifying the particular job.

Following the description of the model execution is the section specifying the algorithm to be used. Multiple algorithms can be specified, and will be executed simultaneously. Parameters for the algorithms are kept to a minimum, are generally intuitive, and provided with sensible default values. In the example shown, eight simultaneous runs will be performed using a parallel implementation of the Nelder-Mead Simplex algorithm, from random starting points. The algorithm description also specifies a desired solution tolerance. For the Simplex algorithm, and the new variant described in this paper, this specifies the convergence criterion by defining the magnitude of the fractional gradient of the final simplex.

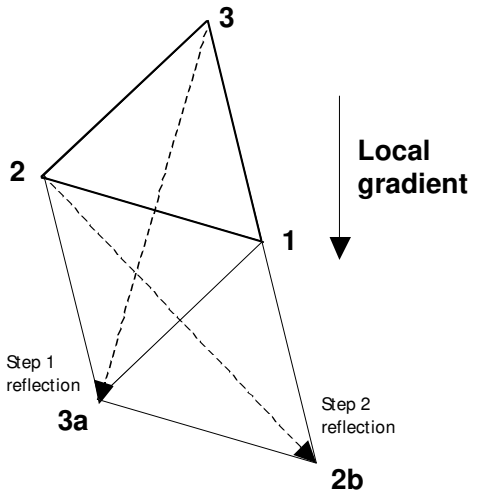


Fig. 2. Sequential Nelder-Mead reflection.

This is sufficient to specify the entire optimization experiment to Nimrod/O. Importantly, the user is not concerned with the computational platform, since support for this is provided by Nimrod or EnFuzion. Issues relating to the Grid are managed through the Nimrod Portal.

3. Reducing Set Concurrent Simplex (RSCS)

The application of supplementary search directions to the Nelder-Mead algorithm, drawing on the methods of the Multidimensional Search (MDS) algorithm of Dennis and Torczon [16–18] has been suggested previously by Hamma [19]. This still implemented additional searches sequentially. As illustrated in Figs 2 and 3, a straightforward, concurrent implementation of all the possible Nelder-Mead search directions is potentially inefficient.

When the search directions of the Nelder-Mead simplex algorithm are applied sequentially, step one producing the new vertex 3a and step 2 producing the new vertex 2b, the search directions are generally downhill, relative to the local gradient. However, if they are applied concurrently, it may quite often be the case that one or more may be to some degree uphill (for example, the trial vertex 2a in Fig. 3). Note: the MDS search directions have been omitted for clarity: they would be from vertices 2 and 3, through vertex 1.

In this paper a different approach to generating supplementary search directions for concurrent search is proposed. The search direction from the worst vertex is through the centroid of the remaining n vertices, as

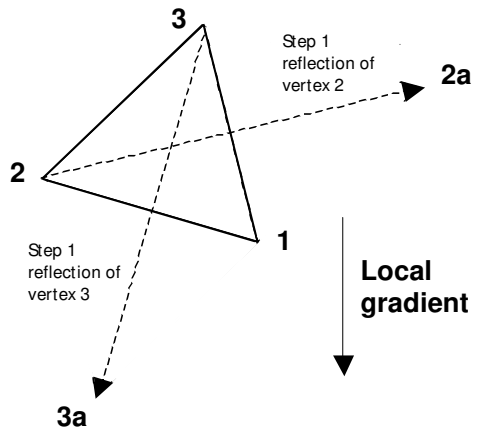


Fig. 3. Concurrent Nelder-Mead reflection.

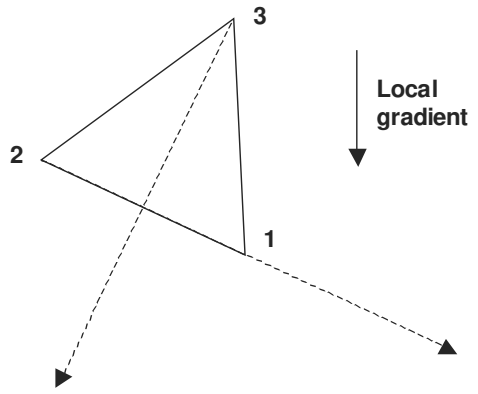


Fig. 4. RSCS search directions.

in the normal Nelder-Mead algorithm. But the search direction from the next worst vertex is through the $n-1$ remaining vertices that are better than it, and so on, until the search direction from the second best vertex is reduced to the MDS search direction though the best vertex. All searches are performed concurrently, and all vertices are independently relocatable.

The method can be considered as deriving from a hybrid of the Nelder-Mead and MDS algorithms. In this work it will be referred to as the Reducing Set Concurrent Simplex (RSCS) algorithm. The set of search directions generated is illustrated in Fig. 4. Comparing the search directions illustrated in Fig. 4 with the concurrent search directions of Fig. 3, it may be noted that the search directions generated by RSCS are more likely to be downhill.

Viewed in two dimensions RSCS can appear to be a simple hybrid of Nelder-Mead and MDS. However, if the 3-dimensional case is considered it becomes apparent there are differences. Figures 5, 6 and 7 show

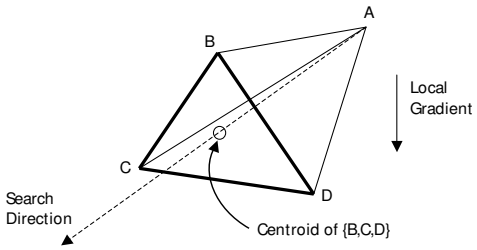


Fig. 5. First RSCS search direction – 3D case.

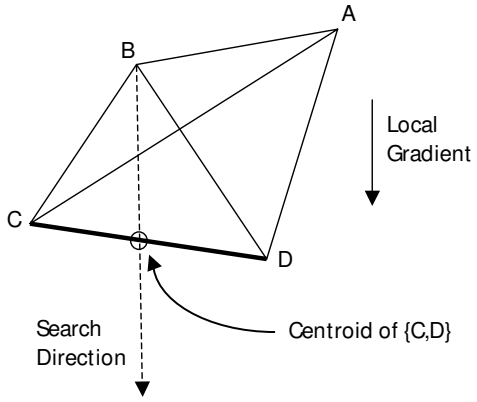


Fig. 6. Second RSCS search direction – 3D case.

the three search directions constructed by RSCS, and the edges and vertices used in their construction. The search direction shown in Fig. 6 will not appear in either the Nelder-Mead or MDS algorithms.

4. Numerical experiments

We have assembled a number of case studies drawn from interesting and challenging scientific and engineering applications. These were used to test and assess the performance of the individual algorithms. The following problems were used for these investigations:

4.1. Laser 1 and 2

A two-dimensional test surface was derived from the computation of a quantum electrodynamical simulation of a laser-atom interaction experiment [10]. The base case, Laser 1, is quite a smooth surface, the dataset containing only 4 minima, of which the global minimum is quite dominant, as can be seen in Fig. 8(a). Additive fractal noise was overlaid on this dataset to develop a "noisier", more challenging surface to test

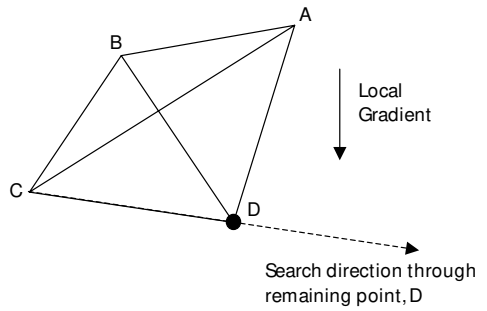


Fig. 7. Third RSCS search direction – 3D case.

the algorithms. This dataset, Laser 2, contained 1157 local minima of varying severity, and is illustrated in Fig. 8(b).

4.2. Crack 1 and 2

Finite element analysis of a thin plate under cyclic loading, with a cutout specified by parameters, was used to generate the Crack datasets [20]. Common practice in damage tolerant design has been to minimise the maximum stress under load. Isosurfaces of these stress values are shown in Fig. 8(c). This dataset, Crack 1, was reasonably smooth, with only 26 local minima. A new approach in modeling stressed components is to attempt to maximize durability. The Crack 2 model seeks to maximise the life of the part as determined by the minimum time taken for fatigue crack growth to a defined length from any of a number of starting crack locations. Isosurfaces at a number of values are shown in Fig. 8(d). In contrast to Crack 1, this dataset was "noisy", with 540 local maxima, and discontinuous isosurfaces.

4.3. Aerofoil

This test case models the aerodynamic properties of a two dimensional aerofoil. The objective function to be minimised is the lift-drag ratio [2], and this is computed by executing a Computational Fluid Dynamics model of the object. Figure 8(e) shows a number of isosurfaces in the parameter space investigated. The dataset was generally smooth, with only 12 local minima and a dominant global minimum.

4.4. Bead

The application from which this case study was drawn used a ceramic bead to minimise distortion of the radiation pattern of a mobile telecommunications handset during testing [21]. The objective function value, derived from an FDTD full-wave analysis of the handset and signal feed cable structure, was a measure of transmission strength through the bead at 1 GHz. The dataset for the Bead case study, of which isosurfaces for a particular value are shown in Fig. 8(f), is quite complex and contains 298 local minima.

4.5. Rosenbrock's function

In order to provide a point of comparison, the well-known Rosenbrock's function in two dimensions was included. The objective function values for this test case were directly computed from:
$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ for $x_i \in [-2, 2]$ which has one local minimum at $f(1, 1) = 0$.

4.6. Case study assembly

It is generally not practical to directly use complex, real-world problems as test cases. The "black box", when queried, can take a considerable amount of time and computational resource to provide a response. So parameter sweeps were made of the numerical models that form the basis of these test cases, and the output data stored. These pre-computed data are interrogated, and linear interpolation employed to provide realistic responses from what are, in effect, "sandboxes" in which optimization programs can readily be tested. These "sandboxes" themselves represent a large investment of time and computational resource – for example, the data acquisition necessary to build the "Bead" test case required over 2 months of continuous computation on a multi-processor supercomputer.

Generally, the case studies fell into 2 sets:

- Smooth, with a dominant global minimum (Laser 1, Crack 1, Aerofoil, Rosenbrock's function)
- Multiple/many local minima, non-convex (Laser 2, Crack 2, Bead)

In formulating ideas about the most appropriate algorithm for use with a particular problem, a great deal may depend on whether the problem encountered is "noisy" or "smooth". Particular attention has been addressed to this issue when assessing algorithm performance.

Table 1
Median results obtained across 10 runs – Objective function values

	Laser 1	Laser 2	Crack 1	Crack 2	Aerofoil	Bead	Rosenbrock
NM	**−0.481**	−0.032	191.9	5299	−67.85	3.66	**0**
MDS	**−0.481**	0.279	**188.0**	**5319**	−67.92	**1.63**	0.066
RSCS	**−0.481**	**−0.286**	193.5	5310	**−68.56**	2.41	0.186

Table 2
Median results obtained across 10 runs – Function evaluations

	Laser 1	Laser 2	Crack 1	Crack 2	Aerofoil	Bead	Rosenbrock
NM	97	100.5	106	66	88	50	240
MDS	96	93	94	260.5	121	49	2358
RSCS	103	120.5	132	180	118	106	227

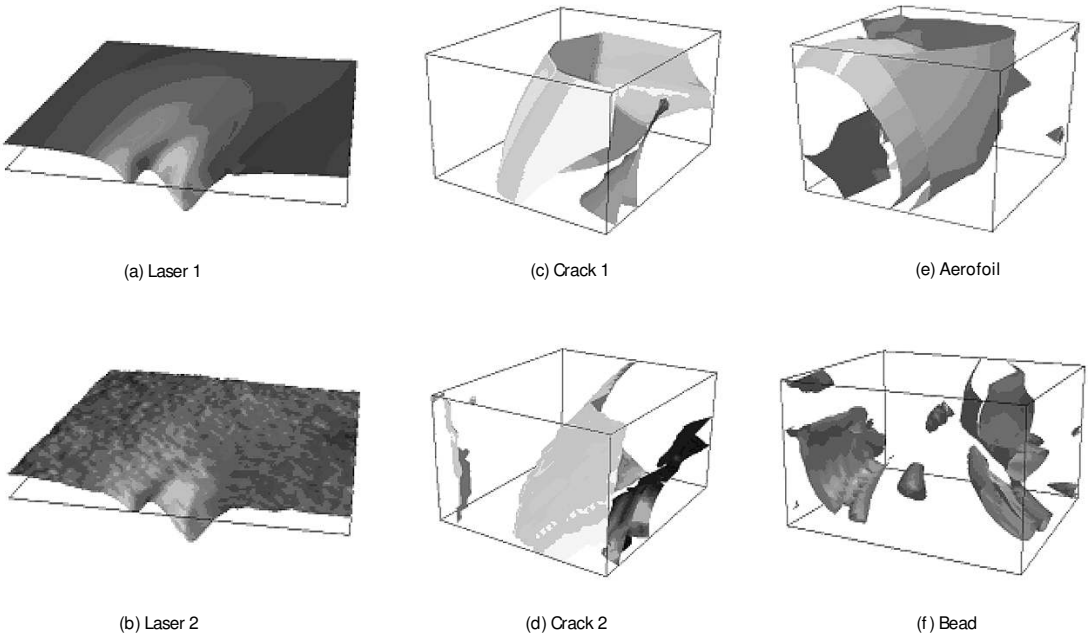


Fig. 8. Test case isosurfaces.

5. Results of Experiments

To evaluate the new RSCS algorithm, it was compared with the original, Nelder-Mead algorithm and an implementation of the MDS algorithm. The Nelder-Mead algorithm performs evaluation of four points when considering relocation of each vertex, corresponding to the various alternatives of reflection, extension and contraction of the simplex. All these points are independent of each other, completely defined by the existing simplex geometry, and could be evaluated concurrently. The algorithm was slightly modified to exploit this concurrency, and all results reported for the "original" algorithm in this section refer to this parallel implementation.

Each of the algorithms was run on each of the test cases from 10 randomly distributed start points. For the purposes of comparison, in a given test case the same set of start points were used for each algorithm. The starting simplices were right simplices aligned with the coordinate axes. By default they were scaled to 10% of the parameter range for each coordinate, as use of reasonably large simplices has been shown to enhance performance [22]. Convergence criterion for most cases was a fractional step-wise gradient of 10^{-3}.

Function evaluations are performed concurrently in batches in Nimrod/O. The batch count can be interpreted as equivalent to Effective Serial Function Evaluations (ESFE), a measure of the wall-clock time taken for completion, providing the machine has enough pro-

Table 3
Median results obtained across 10 runs – Equivalent Serial Function Evaluations

	Laser 1	Laser 2	Crack 1	Crack 2	Aerofoil	Bead	Rosenbrock
NM	24.5	25.5	26.5	16.5	22	12.5	63.5
MDS	16.5	**16**	**11**	29.5	14	**6**	393.5
RSCS	**13.5**	**16**	12	**16**	**10.5**	10	**29**

Table 4
Best objective function values obtained in 10 runs

	Laser 1	Laser 2	Crack 1	Crack 2	Aerofoil	Bead	Rosenbrock
NM	**−0.48**	**−0.56**	**187.6**	5353	**−68.64**	−26.98	**0**
MDS	**−0.48**	**−0.56**	**187.6**	**5357**	**−68.64**	−16.12	3e-4
RSCS	**−0.48**	**−0.56**	**187.6**	5347	**−68.64**	−26.91	**0**

Table 5
Time taken, in ESFE, to achieve best objective function values, across 10 runs

	Laser 1	Laser 2	Crack 1	Crack 2	Aerofoil	Bead	Rosenbrock
NM	24	24	25	23	20	**16**	54
MDS	**16**	**12**	13	1000	14	7	1000
RSCS	22	**12**	**9**	42	**7**	12	**39**

cessors to concurrently evaluate all points, a reasonable assumption given the ready access to cheap clusters.

Analysis of the returned objective function values using the Shapiro-Wilk W test statistic determined that the results are not normally distributed. For this reason, median values and non-parametric, descriptive statistical methods are used for comparison of algorithms.

Tables 1, 2 and 3 show, for each algorithm on each test case over the 10 runs performed, respectively:

- The median objective function value obtained,
- The median number of function evaluations performed (FE) and
- The median Effective Serial Function Evaluations (ESFE).

For each test case, the **best median objective function value** is highlighted in **bold** type. Also highlighted is the **fastest time**, in terms of ESFE, for an algorithm to achieve a result within 10% of the best median objective function, as a percentage of the range of median values obtained.

Table 4 shows the best objective function value obtained in 10 runs for each algorithm on each test case. Table 5 shows the actual ESFEs required to obtain that result. For each test case, the best objective function value obtained by any algorithm, and the fastest ESFE to obtain that value, are highlighted in **bold** type.

From Table 1, it can be seen that RSCS appears to deliver slightly better results than Nelder-Mead on the majority of test cases. From Table 4 it can be seen that RSCS was also capable of equalling the best objective function value returned by both the Nelder-Mead and MDS algorithms on almost all test cases. In the one case it fell slightly behind, it was by less than 0.2%. Analysis of objective function values returned, using the Kruskal-Wallis H test statistic and pair-wise comparisons using Mann-Whitney U test statistics indicated there was no statistically significant difference in the quality of results returned by RSCS, Nelder-Mead simplex and MDS algorithms.

The only significant remaining difference between RSCS and the other algorithms is thus its speed. RSCS gave median ESFE that were consistently better than the Nelder-Mead algorithm. On average, RSCS was 78% faster.

From Table 3, it can be seen that RSCS is approximately 20% faster than MDS on average. This excludes the time taken by MDS on Rosenbrock's function, which was considered a pathological example of the tendency of simplicial methods toward premature convergence [23]. If the median time taken on Rosenbrock's function is included, RSCS is on average faster than MDS by a factor of 4.5. The decision to include these poorer results may be justified by reference to Table 4 in which it can be seen that on two different test cases MDS terminated by exceeding the maximum permissible iterations (1000), rather than satisfying convergence criteria. It can be conjectured that the insistence on congruency of consecutive simplices in the MDS algorithm forces premature contraction of the simplex diameter.

To further investigate the behaviour of the algorithms, the median objective function values achieved after each iteration of each algorithm across all 10 runs

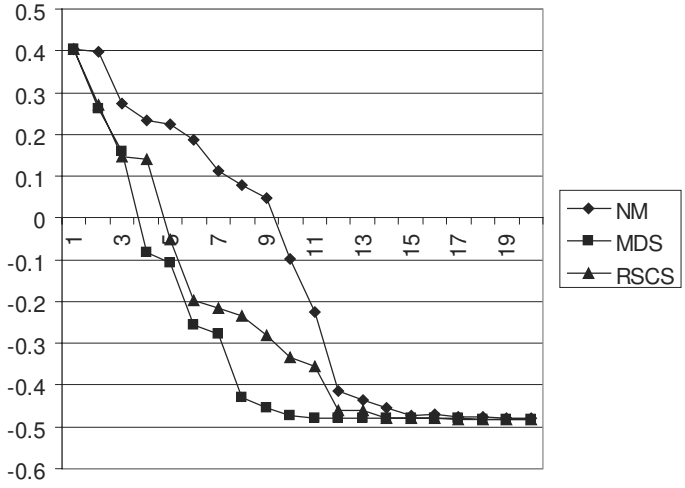


Fig. 9. Convergence history of median values – Laser 1.

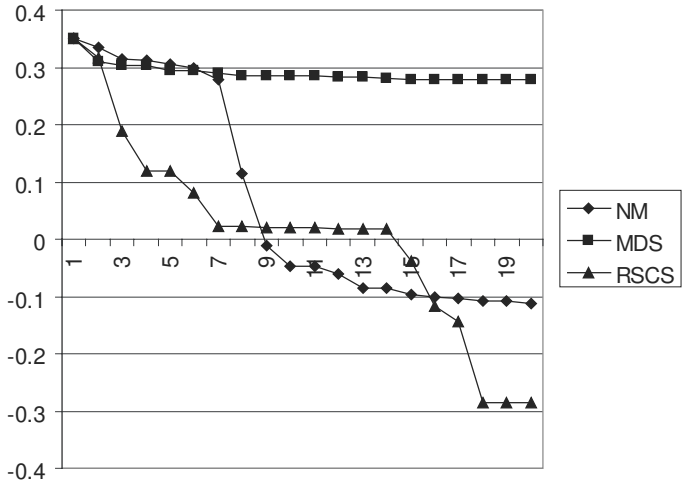


Fig. 10. Convergence history of median values – Laser 2.

on each of the real-world test cases were examined. As noted above, the case studies used can be classified into two classes, "smooth" and "noisy". In Figs 9 and 10 the two case studies derived from the quantum electrodynamical models, Laser 1 and Laser 2, are shown. The first of these exemplifies a "smooth" test case, the latter a "noisy" one.

From Fig. 9 it can be seen that all methods achieve similar results in terms of the objective function values obtained, as can be confirmed by reference to Table 1. Both RSCS and MDS reach better intermediate results faster than Nelder-Mead, and converge slightly earlier, demonstrating the advantages of treating multiple vertices simultaneously.

In Fig. 10, it is MDS that is left behind, as several of the test runs from which the median is drawn become trapped in local minima in regions of low overall gradient far from the global minimum. Nelder-Mead and RSCS obtain better results, with more of the RSCS runs terminating successfully at or near the global minimum. From these brief observations it may be concluded that RSCS is a more consistently reliable method, across

both smooth and noisy test cases.

6. Conclusions

A method of concurrent execution of a simplex optimization algorithm has been proposed, and its performance on a range of real-world problems compared with the popular Nelder-Mead and MDS algorithms from which it is derived. The supplementary search directions used are constructed from reducing sets of vertices in a manner which increases the probability they will be aligned with the local gradient.

In general, the Nelder-Mead and MDS algorithms perform well on the "smooth" test cases, but poorly on the class of cases with noise or many local minima. The new algorithm, RSCS, performs reasonably (or very) well on both types of problems. It generally provides equivalent or slightly better results, and delivers them considerably faster.

A general-purpose optimization toolset, Nimrod/O, in which the prototype and test algorithms have been implemented has also been described. The toolset makes the use of parallel and distributed computing for automatic optimization in the engineering design process readily and easily applicable. Nimrod/O can run a number of very different optimisation algorithms, in parallel if the resources are available. It allows a user to try different algorithms and select those which perform best on a particular problem. The toolset is configured for use on parallel computers, collections of workstations on networks, or grid-based computing resources.

By increasing the speed of the optimization algorithm using parallel computing resources, the method described delivers automatic optimization as a feasible tool for use by the practising engineer tackling real-world problems. With the growing availability of inexpensive computing clusters, rigorous exploration of design alternatives can be made, rather than haphazard, ad hoc evaluation of limited numbers of prototype designs.

References

[1] D. Abramson, A. Lewis and T. Peachey, *Nimrod/O: A tool for Automatic Design Optimization*, in Proceedings of The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, China, 2000.

[2] D. Abramson, A. Lewis and T. Peachey, *Case Studies in Automatic Design Optimisation Using the P-BFGS Algorithm*, in Proceedings of 2001 High Performance Computing Symposium (HPC'01), Seattle, WA, 2001, 104–109.

[3] D. Abramson, A. Lewis, T. Peachey and C. Fletcher, *An Automatic Design Optimization Tool and Its Application to Computational Fluid Dynamics*, in Proceedings of ACM/IEEE SC2001 Conference, Denver, CO, 2001.

[4] R. Fletcher, *Practical Methods of Optimization*, (2nd ed.), John Wiley & Sons, Chichester, 1987.

[5] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981.

[6] T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, NY, 1996.

[7] J. Nelder and R. Mead, A simplex method for function minimization, *Comput. J.* **7** (1965), 308–313.

[8] M. Wright, Direct search methods: Once scorned, now respectable, in: *Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis*, D. Griffiths and G. Watson, eds, Harlow, UK: Addison Wesley Longman, 1995, pp. 191–208.

[9] H. Neddermeijer, G. van Oortmarssen, N. Piersma, R. Dekker and J. Habbema, Adaptive extensions of the Nelder and Mead simplex method for optimisation of stochastic simulation models, Faculty of Economics, Erasmus University, Rotterdam, The Netherlands, Tech. Rep., Econometric Institute Report EI2000-22/A, 2000.

[10] D. Abramson, R. Sosic, J. Giddy and B. Hall, *Nimrod: A Tool for Performing Parametised Simulations Using Distributed Workstations*, in Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

[11] A. Lewis, D. Abramson, R. Sosic and J. Giddy, *Tool-Based Parameterisation: An Application Perspective*, Proceedings of Computational Techniques and Applications Conference (CTAC95), Melbourne, Australia, 1995, 463–469.

[12] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst and N. White, *The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing*, Proceedings of the Australian Computer Science Conference (ACSC97), Sydney, Australia, 1997.

[13] D. Abramson, Nimrod/O, in Nimrod: Tools for Distributed Parametric Modelling, http://www.csse.monash.edu.au/~davida/nimrod/nimrodo.htm, Last viewed 7 November, 2005.

[14] T. Peachey, Nimrod/O user's guide: for version 2.1.x, Monash University, Melbourne, Australia, Tech. Report, 2003.

[15] I. Foster and C. Kesselman, Globus: A metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications* **11** (1997), 115–128.

[16] J. Dennis and V. Torczon, Direct search methods on parallel machines, *SIAM J. Optim.* **1** (1991), 448–474.

[17] V. Torczon, *Multidirectional Search*, Ph.D. dissertation, Rice University, Houston, TX, 1989.

[18] V. Torczon, On the convergence of the multidirectional search algorithm, *SIAM J. Optim.* **1** (1991), 123–145.

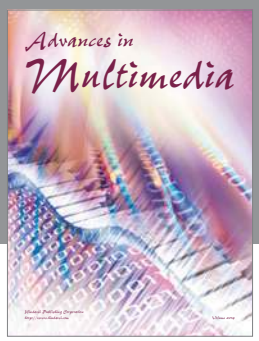
[19] B. Hamma, Local and global behavior of moving polytope algorithms, CERFACS, Toulouse, France, Tech. Report TR/PA/97/39, 1997.

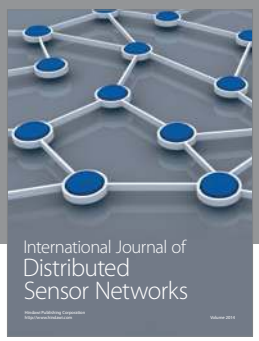
[20] T. Peachey, D. Abramson, A. Lewis and R. Jones, *Distributed Optimization Using Nimrod/O and Its Application to Fault Tolerant Structures*, Fifth International Conference on Parallel Processing and Applied Mathematics (PPAM 2003), Czestochowa, Poland, 2003.

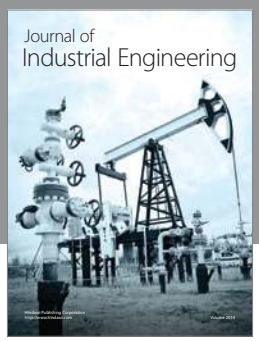
[21] A. Lewis, S. Saario, D. Abramson and J. Lu, *An Application of Optimisation for Passive RF Component Design*, Proceedings of the Conference on Electromagnetic Field Computation, Milwaukee, 2000.

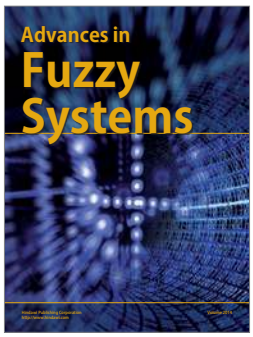
[22] D. Humphrey and J. Wilson, A revised simplex search procedure for stochastic simulation response-surface optimization, in: *Proceedings of the 1998 Winter Simulation Conference*, D. Medeiros, ed., Piscataway, N.J.: IEEE Press, 1998, pp. 751–759.

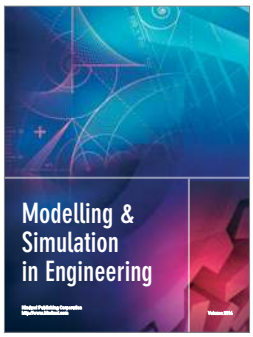
[23] R. Barton and J.J.S. Ivey, Nelder-Mead simplex modifications for simplex optimization, *Management Science* **42** (1996), 954–973.

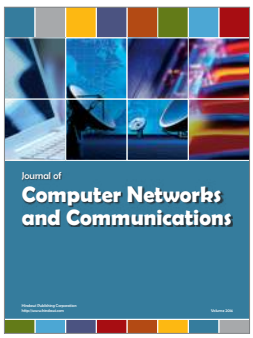












Hindawi
Submit your manuscripts at
http://www.hindawi.com