A Study of Information Sharing in Deterministic Swarms

Ben Stott
Supervised by Andrew Eberhard and Ian Grundy
RMIT University

Introduction

Particle swarms (Kennedy & Eberhart, 1995) are a meta-heuristic popular in computer science used to (hopefully) obtain a global optimal solution for problems that are multimodal or have noisy data (Kennedy & Eberhart, 2001)[1].

Another active research area in deterministic optimisation are derivative free methods based on "frames" and "grids" that try to intelligently probe the local structure of the function via only function values (Price & Coope, 2003). Such methods are provably convergent to local stationary points[2], under some specific criteria.

To date no-one has considered the potential for running multiple copies of direct search algorithms in parallel. Such a scheme would only be of interest if schemes for information sharing, as is done in particle swarms, could be devised. In the case of direct search methods there are numerous classes of information that might be of interest in such a context which include:

a) The formation of different frames, spawning new processes by the sharing of function evaluations, this is much like the generation of descendants in genetic algorithms.

b) The sharing of gradient and curvature information locally between different ("adjacent") realisations.

c) The sharing of information regarding the best solutions found both locally and globally.

A scheme based on c) here was studied, based on how we could best share information about the current solution set among current parallel iterations.

Background

There exists no compelling proof of convergence or mathematical justification for particle swarm optimisers (PSOs). Specifically, a particle swarm can be defined by a set of equations per particle,

$$\begin{aligned}\boldsymbol{v}(t+1,i) = {} & w * \boldsymbol{v}(t,i) \\ & + c_1 * \varphi_1(t) * \big(\boldsymbol{l}(t,i) - \boldsymbol{p}(t,i)\big) + c_2 * \varphi_2(t) * \big(\boldsymbol{g}(t) - \boldsymbol{p}(t,i)\big) \\ \boldsymbol{p}(t+1,i) = {} & \boldsymbol{p}(t,i) + \boldsymbol{v}(t+1,i)\end{aligned}$$

[1] In this paper I will consider that all our optimization problems are cast as minimization problems.
[2] When the function is differentiable and when a suitable replacement for the derivative, called the subderivative, is available for non-smooth functions – however we do not assume that such information is available to the algorithm.

where $\mathbf{v}$ is the velocity of the ith particle at time t, w is termed an *inertial factor* (a constant that defines how likely a particle is to follow its previous flight path), $\boldsymbol{l}(t, i)$ defines the *locally best* function position of particle i at time t, $\boldsymbol{g}(t)$ is the globally best function position of any particle, c_i are weighting constants towards the local and global best values and $\varphi_i(t)$ is a random value in [0, 1), randomly chosen at each time step.

Note that the set of information links between each particle and its neighbour builds a graph (a communication network) termed the topology of the swarm. A commonly used topology is the ring, in which each particle has just two neighbours, but there are many other topologies, including the "null" topology in which no node is connected to any other node. Note that a topology can be defined in terms of neighbours in the search space, since if we imagine each particle to have a label we are perfectly free to relabel these arbitrarily so they suit our scheme. This is usually the way we can modify the local best value $\boldsymbol{l}(t, i)$; we choose our topology arbitrarily and simply take all nodes that are one hop away from node i as the set of local nodes.

It should be clear from this definition of velocity and position, then, that a particle has no concept of a globally optimum value. However, as has been demonstrated in (Schutte & Groenwold, 2005), particle swarms are able to perform to an admirable standard with minor tweaks to the inertial weights.

The reasons for their success is debated as to whether there is a balance between exploratory behaviour, that is, searching a broader region of the search-space, and exploitative behaviour, that is, a locally oriented search so as to get closer to a (possibly local) optimum. Moreover the success of these methods is greatly influenced by "parameter tuning" leading to a machine learning justification for these methods, in an effort to find the best sequence of both inertial weights and constant biases.

PSOs have seen a large body of recent work dedicated to them, however, because of their trivial ability to be parallelised. It should be immediately obvious that for the case of the null topology, local and global best updates can be done in parallel, allowing for the query of each particle at the same time.

From a mathematical perspective, direct search methods *do* come with a convergence guarantee. Direct-search methods define a set of search directions in which to sample the search-space around the current location. They decide how far to move in each direction (or whether to move away from the current location) based on the success or failure of finding a better objective function value than that at the previous iteration. If we specifically focus on frame-based methods (which can be generalised to a grid if need be), these frames are constructed from *positive bases*[3], in a way that a specific polling order can guarantee convergence; for more information see (Coope & Price, 2001)

Works on parallelisation of these methods have shown that much potential exists for distribution of load in parallel or cloud computing environments (Macklem, 2009). Information in a frame can also be used to estimate gradient and curvature information. This allows the importation of methods from numerical quasi-Newton optimisation to use sophisticated estimation techniques for the calculation of "good" descent directions.

[3] From a regular set of basis vectors one can construct a positive basis via the introduction of at least one extra linearly dependent vector so as to be able to represent any vector in the space as a linear combination using only positive coefficients.

For a study in the presence of sampling of sub-gradients see (Hudson, 2007), and using only function values see (Coope & Hutchinson, 2009).

A New Approach

The paper I found most useful in my quest to create a deterministic swarm was (Audet & Dennis, 2006); within they propose a direct-search method wherein a mesh around a candidate solution is built, and their algorithm queries points along this mesh. They termed this algorithm *mesh-adaptive direct search*, or MADS for short. If the current candidate solution is lower than all the points queried on the mesh (i.e. the candidate solution forms an analogue of a *quasi-minimal frame*, see page 160 of (Conn, et al., 2009)), the size of the mesh is decreased, while if any of the query points on the mesh gain *sufficient descent* (i.e. a substantial improvement over the candidate solution), the mesh centre is moved and the size of the mesh is increased. It should be quite easy to see that the sequence of *unique* mesh centres formed will be finite; one can imagine a local basin where the mesh starts anywhere 'up' the basin; while the mesh size might expand to eventually fall outside the basin, these points will only be considered by the mesh if they gain sufficient descent over the current centre, otherwise the mesh will be shrunk to an infinitesimal size (though in practice this is often "chosen" through the use of an epsilon mesh tolerance, especially given the architecture of modern computers). However, note the bolding of "unique", above – if we examine the sequence of all mesh centres, its cardinality will be (countably) infinite, which is what necessitates the use of an epsilon.

So far we have discussed a vanilla version of MADS. How can we now integrate particle swam methodologies into this framework?

Note the point about sufficient descent; if we gain sufficient descent we move the centre of our mesh. This should now immediately lead to an obvious way to create a hybrid swarm and mesh optimiser: since the swarm will terminate in finite time, we can simply use this as an additional step to locate a point of sufficient descent. This won't disrupt the fact that the sequence of iterates converges on to a local minima; it is ostensibly the same as if we just started our locally-convergent algorithm from this new point originally. We will term this Optimisation Algorithm 1 or OA1 for short.

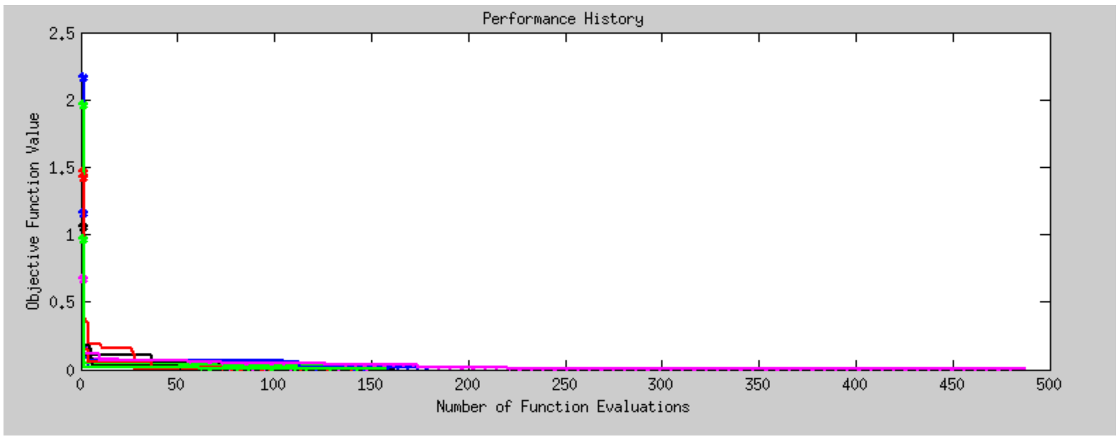

Figure 1: 10 runs of OA 1

Unfortunately this algorithm performs rather poorly. Out of approximately 30 runs (only ten of which are shown in figure 1 due to limitations of the software used) that attempted to find the global minimum of Griewangk's function, defined by

$$f_n(x_1, \dots, x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} \mathrm{x}_i^2 - \prod_{i=1}^{n} \cos\frac{x_i}{\sqrt{i}}$$

(using n = 2) only 1 out of the 30 runs found the correct optimum point (at $\mathbf{0}$), taking nearly 500 function evaluations to do so. This is particularly poor performance both with the aim towards global convergence and minimising the number of function evaluations; most other iterations of this algorithm were taking approximately 150 evaluations to come to a value that was "close" to the optimum point, and comparable "optimum" values can be found with a vanilla PSO with fewer function evaluations.

Fortunately, there are two observations to be made here: First, note that we haven't yet utilized the ability to parallelise, aside from the implicit parallelisation of the swarm and the mesh individually. It would be optimal, however, to attempt to parallelise these together. To achieve this, we take the total number of computing cores available to us and split them into one of either two pools. The first pool runs purely PSOs, while the second pool runs purely MADS (note that due to an imposed restriction by MATLAB, our research was only able to *simulate* this behaviour through the use of a queue; the PSO would produce a point into the queue and then the MADS would take a point out of the queue after the PSO had terminated, but this restriction shouldn't exist in, say, a C implementation of this algorithm). By carefully maintaining a global cache of points seen so far, the PSO is able to avoid neighbourhoods of previously-seen points (being reinitialised if it comes within a Euclidean norm of *k* of a previously seen point) and thus produce only "new" basins for the MADS to optimise within. This produced some promising results; we noted now about 15% of the runs of our algorithm would converge on the global optimum, however this was far more potent with the second observation.

The second observation relied upon noting that the MADS algorithm was being "moved" out of a good basin due to the nature of the sufficient descent criteria. If the PSO finds a point of sufficient descent in another basin, we move our mesh without regard for the current basin we're exploring. It could (and did) so happen that we're almost at the top of our current basin (which, for the sake of thought experiment, we'll claim to be the global optimum) and the PSO has found a point near the bottom of a secondary basin (which is only locally optimum). This severely hampers efforts to find a global optima, however doesn't break the algorithm's promise of a local optima.

This could, however, be repaired by a trivial modification to the PSO equation:

$$\begin{aligned}\boldsymbol{v}(t+1,i) = {} & w * \boldsymbol{v}(t,\, i) \\ & + c_1 * \varphi_1(t) * \big(\boldsymbol{l}(t,i) - \boldsymbol{p}(t,\, i)\big) + c_2 * \varphi_2(t) * \big(\boldsymbol{g}(t) - \boldsymbol{p}(t,i)\big) \\ \boldsymbol{p}(t+1,i) = {} & \boldsymbol{p}(t,i) + \boldsymbol{v}(t+1,i) \\ \boldsymbol{p}(t+1,i) = {} & \min(\boldsymbol{p}(t+1,i), \mathrm{NELDERMEAD}(\boldsymbol{p}(t+1,i), 10))\end{aligned}$$

where $\mathrm{NELDERMEAD}(\boldsymbol{p}, k)$ defines k steps of the Nelder-Mead algorithm starting with the point $\boldsymbol{p}$ on the simplex (this algorithm is well described in (Nelder & Mead, 1965)). The aim of this additional step isn't to replace the MADS step, thus the NELDERMEAD algorithm doesn't need to be run to completion; we seek to only be more confident that when we move our mesh, we've moving it to a better basin. Thus, we made the rather arbitrary decision that this algorithm would only be allowed 10 steps of the Nelder-Mead algorithm (and is the second parameter to this algorithm) – this number could be conceivably higher or lower and it wasn't investigated too thoroughly.

These two observations lead to a significantly improved algorithm (OA2), as seen in figure 2. Note that the number of function evaluations in figure 2 is artificially low, as I ran out of time to fix this issue. It should be increased by a factor of 10; thus it was taking on average 1300 function evaluations to find the global optimum, instead of 130. However, somewhat more importantly, it finds the global minimum 100% of the time (only 10 iterations shown, again)

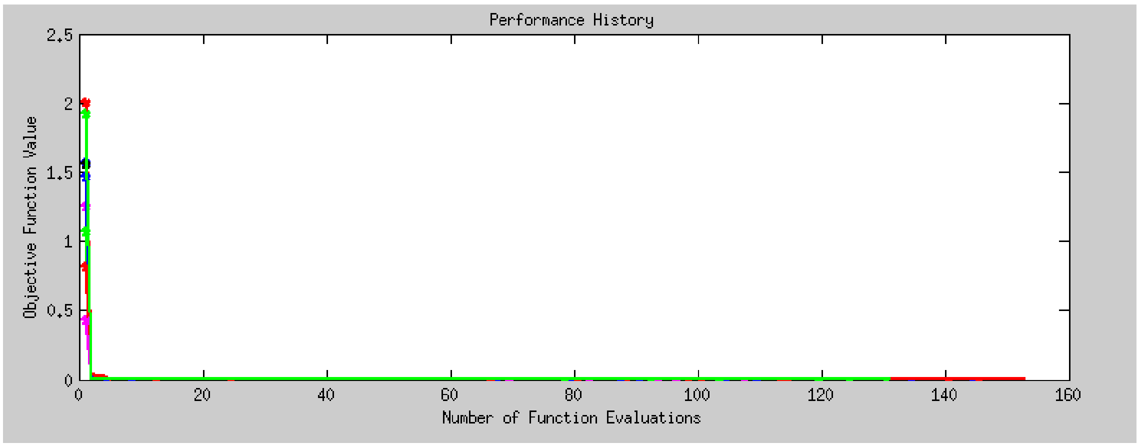

Figure 2: 10 runs of OA2, all of which find the global minimum on the Griewank function

For comparison, using (Grundy & Stacey, 2008), we note that they had a 94% success rate for finding the global optimum on Griewangk's function. My observations were that OA2 had a 100% success rate with significantly less time needed to execute the algorithm: 10 runs of the algorithm presented in (Grundy & Stacey, 2008) (implemented in MATLAB) took in the order of ten minutes to execute, while 10 runs of OA2 (also implemented in MATLAB) took in the order of eight – a 20% improvement.

Testing on the Ackley function, given by

$$f_n(x_1, \dots, x_n) = -20 * e^{-0.2 * \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}} + 20 + e$$

this time with n = 10, we find again a 100% strike rate (seen in figure 3, similar to other figures so far). Again comparing this to (Grundy & Stacey, 2008), we see that their

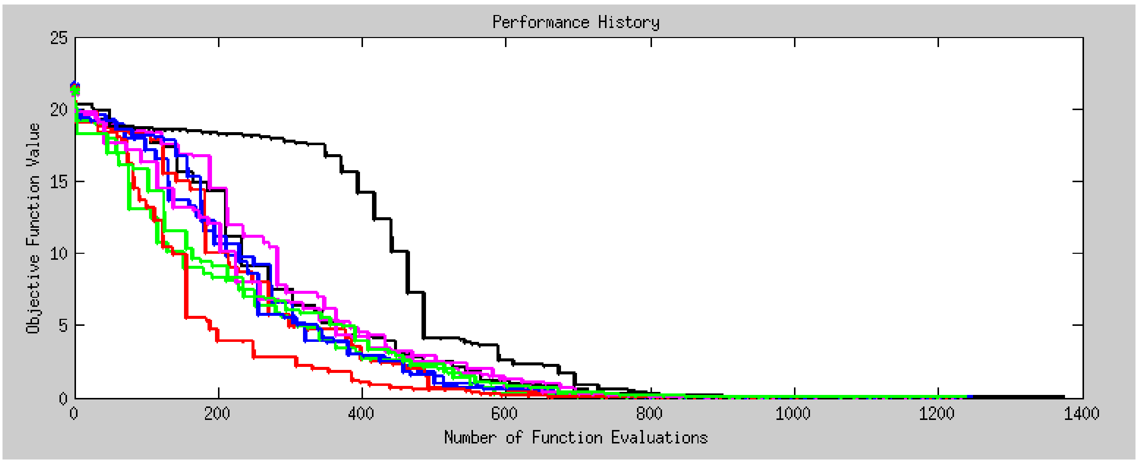

Figure 3: 10 runs of OA on the Ackley function

algorithm performs slightly better than this; both have the same success rate, though a specific version of their algorithm completes this task with just over a quarter of the iterations required for OA2 (note that for now the number of evaluations in our figure needs to be multiplied by 10). However, this is offset by the fact that our algorithm has provable local convergence, whereas the algorithm presented in (Grundy & Stacey, 2008) was manually terminated when it approaches the “correct” point (since this point is known in advance for the Ackley function), and OA2 also comes with slightly stronger termination criteria by default (the mesh epsilon of 10^{-4} means most x values are within 10^{-3} instead of 10^{-2} from the optimum point). Of course, the algorithm given by (Grundy & Stacey, 2008) could be terminated when, say, particles have clumped together, but this will change the reported number of function values and so this report can’t make a comparison with that termination criteria.

Future work

Unfortunately, we ran out of time over the summer to achieve all that we wanted to. One thing that this algorithm seems to lend itself particularly nicely to is integer programming; on a naive test on both a discretised Griewank function and discretised Ackley function (given by the same equation, except each x value was rounded (up) to the nearest integer) with the mesh forced to an integer size, we saw promising results. This work was aimed to be extended to trial problems from MIPLIB, however time constraints and a lack of a specification file for the MIPLIB format meant we were unable to finish this in time. It will, however, remain a useful area of future work, especially if the same global optimum locating ability exists.

We also intend to explore derivative-like calculation similar to (Coope & Hutchinson, 2008), in an effort to order points by priority and cut down on the number of function evaluations if a “better” point is found.

Finally, thorough test function suites for optimisation exist, from De Jong’s test suite to extended lists of functions such as (Pohlheim, 2006), and future work should make use of these functions (as well as functions for an integer-based optimisation algorithm)

Conclusion

There is plenty of room for work on hybridising algorithms from computer science-based optimisation and mathematical optimisation. This project has demonstrated that at least this algorithm is able to provide sufficiently intriguing results that it warrants more research. It is also able to exploit a simplified version of parallelisation that should be sufficient as a proof of concept to show that “difficult” optimisation problems no longer require supercomputers; instead this task is now just about within the realm of personal computers, making it possible for anyone to be able to solve optimisation problems without large banks of domain-specific knowledge.

Thanks must go to both my supervisors, Andrew Eberhard and Ian Grundy, for their patience with me and ideas for work. I also thank both AMSI and CSIRO for the wonderful experiences granted both through the summer research program and the Big Day In, where a large number of talented students presented their summer’s work in a fun environment.

References

Audet, C. & Dennis, J. E., 2006. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.,* 17(2), pp. 188-217.

Conn, A. R., Scheinberg, K. & Vicente, L. N., 2009. *Introduction to Derivative-Free Optimization.* 1st ed. Philadelphia: Society for Industrial and Applied Mathematics and the Mathemtical Programming Society.

Coope, I. D. & Price, C. J., 2001. On the convergence of grid-based methods for unconstrained optimization. *Journal of Optimization,* 11(4), pp. 859-869.

Coope, I. & Hutchinson, D., 2008. *Direction-set updates for derivative-free optimization.* Christchurch, ANZMC.

Coope, I. & Hutchinson, D., 2009. Direction-set updates for derivative-free optimization. *Advanced Modeling and Optimization,* 11(1), pp. 33-41.

Grundy, I. & Stacey, A., 2008. Particle swarm optimization with combined mutation and hill climbing. *Complexity International,* Volume 12.

Hudson, J., 2007. Trust Region Methods Applied to a Newton Bundle Method for Nonsmooth Optimisation. *Honours Thesis, School of Mathematical and Geospatial Sciences, RMIT University, Melbourne.*

Kennedy, J. & Eberhart, R., 1995. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks,* pp. 1942-1948.

Kennedy, J. & Eberhart, R., 2001. *Swarm Intelligence.* s.l.:s.n.

Macklem, M., 2009. Low Dimensional Curvature Methods in Derivative Free Optimization on Shared Computer Networks. *PhD Thesis, Dalhousie University, Halifax, Nova Scotia.*

Nelder, J. A. & Mead, R., 1965. A simplex method for function minimization. *The Computer Journal,* 7(4), p. 308–313.

Pohlheim, H., 2006. *GEATbx: Parametric optimization.* [Online]
Available at: http://www.geatbx.com/docu/fcnindex-01.html
[Accessed 2013].

Price, C. & Coope, I., 2003. Frame-Based Ray Search Algorithms in Unconstrained Optimization. *Journal of Optimization Theory and Applications,* 116(2), pp. 359-377.

Schutte, J. F. & Groenwold, A. A., 2005. A Study of Global Optimization Using Particle Swarms. *Journal of Global Optimization,* 31(1), pp. 93-108.