Asynchronous Multiple Objective Particle Swarm Optimisation in Unreliable Distributed Environments

Author

Scriven, Ian, Ireland, David, Lewis, Andrew, Mostaghim, Sanaz, Branke, Juergen

Published

2008

Conference Title

2008 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-8

DOI

10.1109/CEC.2008.4631130

Asynchronous Multiple Objective Particle Swarm Optimisation in Unreliable Distributed Environments

Ian Scriven, David Ireland, Andrew Lewis, Sanaz Mostaghim and Juergen Branke

Abstract— **This paper examines the performance characteristics of both asynchronous and synchronous parallel particle swarm optimisation algorithms in heterogeneous, fault-prone environments. Algorithm convergence is measured as a function of both iterations completed and time elapsed, allowing the two particle update mechanisms to be comprehensively evaluated and compared in such an environment. Asynchronous particle updates are shown to negatively impact the convergence speed in regards to iterations completed, however the increased parallel efficiency of the asynchronous model appears to counter this performance reduction, ensuring the asynchronous update mechanism performs comparably to the synchronous mechanism in fault-free environments. When faults are introduced, the synchronous update method is shown to suffer significant performance drops, suggesting that at least partly asynchronous algorithms should be used in real-world environments where faults can regularly occur.**

I. Introduction

Particle swarm optimisation (PSO) is fast being established as an efficient optimisation technique for both single and multiple objective design problems. It has been applied to a wide variety of engineering problems, from biomechanics [1] to avionics [2]. Such real-world problems usually require complex, time-consuming computer simulations to solve potential solutions, and as such parallelisation of optimisation algorithms is becoming increasingly prevalent.

The majority of parallel particle swarm implementations are based on the synchronous model, where the optimisation algorithm waits at the end of each iteration until all particle solutions have been returned before updating particle velocities and positions (known as a Parallel Synchronous Particle Swarm Optimisation algorithm, or PSPSO) [3]. This approach can work quite well, provided a number of conditions are met [1]:

1) The optimisation algorithm has uninterrupted access to a homogeneous computer cluster..
2) The analysis function can be evaluated in roughly constant time, regardless of the input parameters.
3) The number of particles in the swarm can be evenly divided by the number of available nodes.

However, it is often difficult (or even impossible) to obtain these ideal operating conditions, and when this happens, the parallel efficiency of the algorithm drops. This drop in parallel efficiency can be mitigated somewhat through the use of asynchronous updates [1], [4]. In a Parallel Asynchronous Particle Swarm Optimisation (PAPSO) algorithm, the algorithm does not wait for all solutions to be returned before updating the velocity and position of solved particles and re-evaluating them. These algorithms can display varying degrees of synchronous behaviour, ranging from 1% synchronous (or pure asynchronous) where particles are updated and submitted for re-evaluation as soon as they are solved, to 100%, at which point the algorithm becomes fully synchronous once more, waiting for all particles to be returned before updates are carried out. While asynchronous updates in sequential particle swarm optimisation algorithms have been well investigated [5], and are generally advantageous resulting in faster convergence to the optimal solution (or optimal set of solutions), these results can not be expected to carry over into the PAPSO algorithm [1]. At the time of writing, the only studies on PAPSO algorithms have focused mainly on parallel efficiency [1], [4], without thoroughly examining the impact of parallel updates on algorithm convergence time, and without looking at the impact of evaluation failure. Failure is an important factor to consider, as it is practically unavoidable in most environments, whether caused by inter-note communication breakdowns, node failure, or even failure of the solver itself. This paper examines the effects of asynchronous updates in PAPSO algorithms in fault-prone environments, and compares the convergence characteristics of PSPSO and PAPSO algorithms against both iteration count and wall-clock execution time.

Ian Scriven and David Ireland are with the School of Engineering, Griffith University, Brisbane, Queensland, Australia (email: {I.Scriven, D.Ireland}@griffith.edu.au).

Andrew Lewis is with the School of Information and Communication Technology, Griffith University, Brisbane, Queensland, Australia (email: A.Lewis@griffith.edu.au).

Sanaz Mostaghim and Juergen Branke are with Institute AIFB, University of Karlsruhe, Germany (email: {mostaghim, branke}@aifb.uni-karlsruhe.de

II. Distributed Particle Swarm Optimisation

Particle swarm optimisation, like most stochastic optimisation algorithms, can be easily implemented in a parallel distributed computing environment. The simplest and most common distributed particle swarm optimisation algorithms utilise a master-slave architecture, where a single controlling (master) processor runs only the optimisation algorithm, and utilises on external (slave) processors to compute potential solutions [6].

The PSO algorithm is based on swarm intelligence, and artificial intelligence technique that studies collective behaviour in decentralised, self-organised groups [7]. It utilises a population of potential solutions, or particles, which move around the design space with every iteration. The movement

of these particles is governed by two equations:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \quad (1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2)$$

Here, $v_{ij}(t)$ is the velocity of particle *i* in dimension *j* at time *t* and $x_{ij}(t)$ is the position of particle *i* in dimension *j* at time *t*. The velocity of a particle depends on both the best position that particle has found to time *t*, $y_{ij}(t)$, and the best solution the entire swarm has found has found to time time *t*, $\hat{y}_{ij}(t)$. The inertial component is scaled by constant *w*, and c_1 and c_2 are constants, usually defined between 0.5 and 3.0, used to control the impact of the local and global components in velocity equation (1). The vector *r* is a vector of random numbers evenly distributed between zero and one generated for each particle at each time step *t*. Once new particle velocities have been calculated, the position of each particle is updated as in (2).

In a synchronous particle swarm algorithm, the algorithm will wait for all particles to be solved before updating particle velocities and positions using equations (1) and (2). This allows the term 'iteration' to be easily defined as one step of the algorithm in which all particles are evaluated. In asynchronous particle swarms, however, the algorithm will wait for only a certain proportion of the particle solutions to be returned before updates are carried out, making an iteration harder to define. For the purposes of comparing PSPSO and PAPSO convergence against iterations completed, a single iteration of a PAPSO algorithm will be defined as a period in which the number of evaluations performed equals the number of particles in the swarm.

The performance of PSPSO algorithms will be seriously impacted if the homogeneous requirement mentioned in section I is not met. If some particles take longer to computer than others, at the end of each iteration most nodes in the cluster will be idle, reducing parallel efficiency, and increasing the execution time of the algorithm. Worse still, if one or more particle solutions are not returned in a timely fashion (in which case they can be said to have failed), those particles must be evaluated again, further decreasing parallel efficiency and negatively impacting the time taken to arrive at the optimal solution. PAPSO algorithms will not suffer as significantly from these issues. A 1% synchronous (i.e. fully asynchronous) PAPSO will theoretically obtain near 100% parallel efficiency, and the impact of failures will be lessened, as a failed particle will not delay any others. A 50% synchronous PAPSO will still have a somewhat higher parallel efficiency than the PSPSO, and will only be impacted by failures if half or more of the particles fail to evaluate.

However, it can be expected that the PAPSO will have some disadvantages compared to the PSPSO in regards to convergence against iterations completed. Whereas the sequential asynchronous PSO yielded convergence improvements over the synchronous PSO, the PAPSO will have somewhat decrease performance. The cause of this becomes obvious when examining the amount of information (previous solutions) available when any given particle is updated. In the sequential asynchronous PSO, the number of previous solutions PS_{sa} for particle *j* at iteration *i* for a swarm of size *N* is given by (3).

$$PS_{sa} = (i-1)\times N + (j-1) \quad (3)$$

For example, on the second ($i = 2$) iteration of a $N = 100$ particle swarm, particle $j = 50$ will have 149 previous solutions available to it. This is more than would be available to a synchronous PSO, which would only have the 100 solutions from the previous iteration available for the same example, as given by (4).

$$PS_{ps} = (i-1)\times N \quad (4)$$

Calculating the amount of information available to a purely asynchronous PSO algorithm is slightly more complex, being based somewhat on when the particle evaluation is completed, but for a particle having around average evaluation time, it can be approximated as (5).

$$PS_{pa} \simeq (i-1)\times N - \frac{N}{2} \quad (5)$$

Unlike the sequential APSO and the synchronous PSO, the PAPSO at iteration $i = 2$ will not usually have all the information from the previous iteration available to it, as many particles will still be under evaluation. As such, for the example used previously, a particles in the PAPSO on the second iteration will only have, on average, 50 previous solutions available to them. This clearly suggests that, in terms of iterations completed, the PAPSO should converge more slowly than the PSPSO. It is expected, however, that this performance decrease will be countered somewhat by the increase parallel efficiency of the PAPSO, which will allow particle evaluations (and thus iterations) to be completed more quickly than in the PSPSO.

III. Simulation Setup and Testing Procedure

In order to examine the performance of the PAPSO and PSPSO in heterogeneous, fault-prone conditions, a distributed particle swarm simulation environment was constructed using both 100 particles and 100 computational nodes. A *gbest* multiple objective PSO algorithm was used, with $y_{ij}(t)$ and $\hat{y}_{ij}(y)$ being chosen at random from the non-dominated sets of solutions found by the entire swarm and by the individual particles, respectively.

Particle evaluation times were controlled so as to conform to a normal distribution, with a mean execution time of $\mu = 5$ seconds and a standard deviation $\sigma = 1$ second. Modelling the evaluation times in this way meant that failures could be detected by monitoring the time taken by each solver. If a solution was not returned with $\mu + 4\sigma$ seconds, it was assumed that the solver had failed. This is a safe assumption, as (from the properties of the normal distribution) 99.997% of evaluations would have completed in this allowed time.

The solver used was an analytical test function consisting of the three functions f_1, g and h and is of the form [8]:

$$\begin{aligned} \text{Minimise } \mathbf{t}(\mathbf{x}) &= (f_1(x_1), f_2(\mathbf{x})) \\ \text{subject to } f_2(\mathbf{x}) &= g(x_2, ..., x_n).h(f_1(x_1), g(x_2, ..., x_n)) \\ \text{where } \mathbf{x} &= (x_1, ...x_n) \end{aligned} \tag{6}$$

This test function has a convex Pareto-optimal front given by (7)

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_n) &= 1 + 9.(\sum_{i=2}^{n} x_i)/(n-1) \\ h(f_1, g) &= 1 - \sqrt{f_1/g} \end{aligned} \tag{7}$$

where $n = 30$ and $x_i \in [0, 1]$. The Pareto-optimal front is formed with $g = 1$ [8].

Simulations were run for varying degrees of synchronous behaviour, ranging from 1% (fully asynchronous) through to 100% (fully synchronous), for failure rates of 0%, 5%, 10% and 20%. Results were averaged over fifteen runs to ensure reliable, repeatable experimental data was obtained. A convergence factor metric was used which measured area of the potential solution space dominated by an approximation to the Pareto-optimal front as a ratio of the solution space dominated by the actual Pareto-front. An approximation to the Pareto-optimal front that is equal to the actual Pareto-front has a convergence factor of one, and a solution set that has not converged will have a convergence factor less than one (but greater than zero). This convergence factor was monitored as a function of both iterations completed and time elapsed.

IV. Results

The first test carried out involved measuring the convergence of the algorithms as a function of iterations completed in a fault-free environment, in order to confirm the previously stated hypothesis that the more asynchronous algorithm variations will take more iterations to converge to the same point as the less asynchronous algorithms. The average convergence factor for 1% synchronous, 30% synchronous, 70% synchronous and 100% synchronous particle swarm algorithms over 100 iterations is shown in Fig. 1.

These results support the stated hypothesis, clearly showing that the fully synchronous algorithm (the PSPSO) converges faster with respect to iterations completed than the asynchronous (PAPSO) variations, with the purely asynchronous algorithm performing the worst of all, as expected. The more synchronous the algorithm is, the faster it converges with respect to iterations completed.

Using more asynchronous updates does, however, increase the parallel efficiency of the algorithm, reducing the time taken for each algorithm iteration. Fig. 2. shows the same set of results, this time plotted as a function of execution time. This graph suggests that for this solver function, the increased parallel efficiency of the asynchronous algorithms

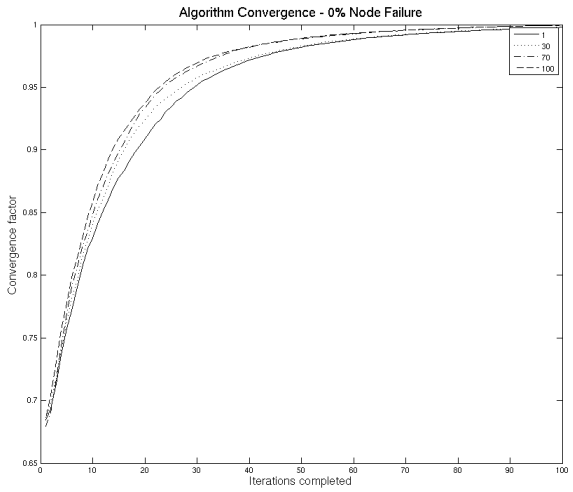

Fig. 1. Algorithm convergence for varying degrees of synchronous behaviour - 1% (fully asynchronous), 30%, 70% and 100% (fully synchronous) - as a function of iterations completed, with 0% failure rate.

mitigates the slowdown in iteration-wise convergence, resulting in similar performance vs. time characteristics as the synchronous algorithm. In fact, the performance (over time) of the fully asynchronous algorithm (1%) and the full synchronous algorithm (100%) are almost identical. The small variations between the different versions of the algorithm can be accounted for by the Gaussian random number generator used to determine the solver execution delays at runtime.

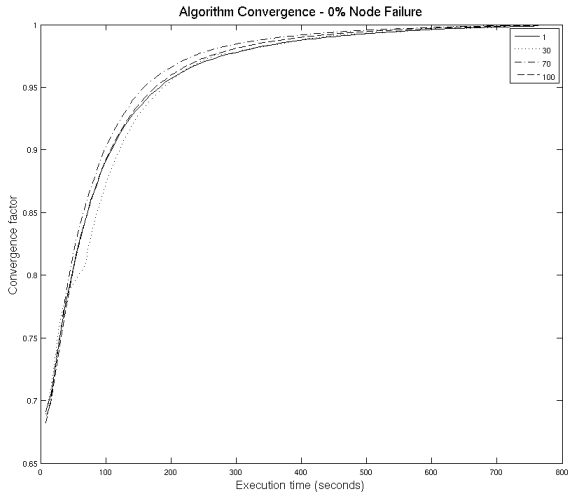

Fig. 2. Algorithm convergence for varying degrees of synchronous behaviour - 1% (fully asynchronous), 30%, 70% and 100% (fully synchronous) - as a function of algorithm execution time, with 0% failure rate.

Fig. 3. shows the impact of faults on the same four algorithm variations. The introduction of faults can be seen to have very little effect on the asynchronous (1%, 30% and 70%) algorithms, whereas the performance of the syn-

chronous algorithm drops markedly. With a 5% probability of failure, the synchronous algorithm must wait for approximately twice as long at each iteration, as failed solvers have to be re-evaluated.

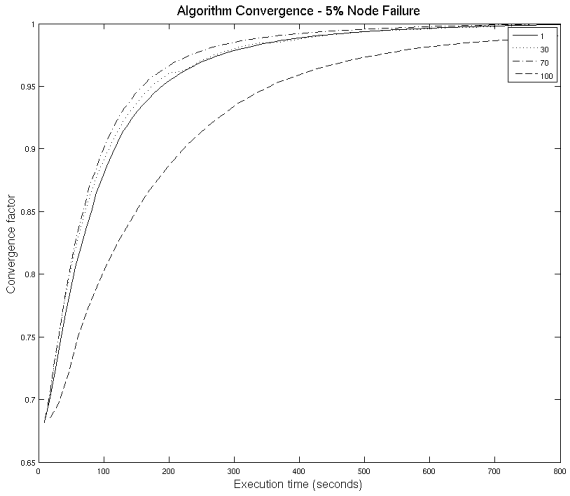

Fig. 3. Algorithm convergence for varying degrees of synchronous behaviour - 1% (fully asynchronous), 30%, 70% and 100% (fully synchronous) - as a function of algorithm execution time, with 5% failure rate.

To more closely examine the effect of the probability of failure on the more synchronous algorithms, a different set of algorithms was used, again including the pure synchronous PSPSO as well as the 70%, 80% and 90% synchronous PAPSO algorithms. For a baseline measurement, the average convergence achieved over time of these algorithms without solver failures can be found in Fig. 4, which again shows the different algorithms performing almost identically.

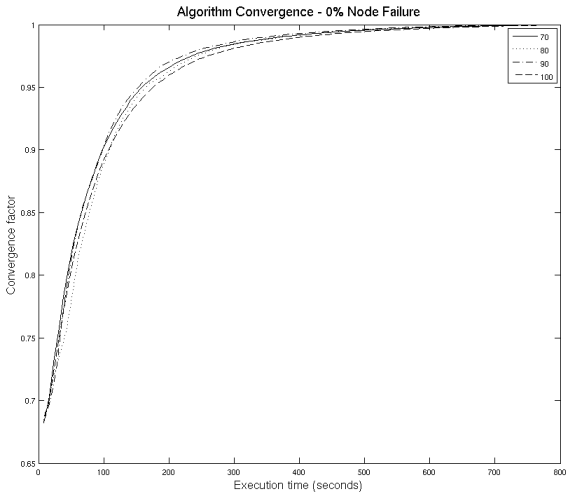

Fig. 4. Algorithm convergence for low degrees of synchronous behaviour - 70%, 80%, 90% and 100% (fully synchronous) - as a function of algorithm execution time, with 0% failure rate.

Figures 5, 6 and 7 show the same algorithm performance measure for a probability of failure of 5%, 10% and 20% respectively. For a failure rate of 5%, again only the purely synchronous algorithm's performance is impacted, however, once the failure rate reaches 10%, the 90% synchronous PAPSO's performance also starts to degrade. Similarly, for a 20% chance of solver failure, the 80% synchronous PAPSO begins to suffer, and the performance of the PSPSO can be seen to degrade even further. This extra drop in the PSPSO's performance can be attributed to particles failing multiple times in the same iteration, further increasing the time required for each iteration.

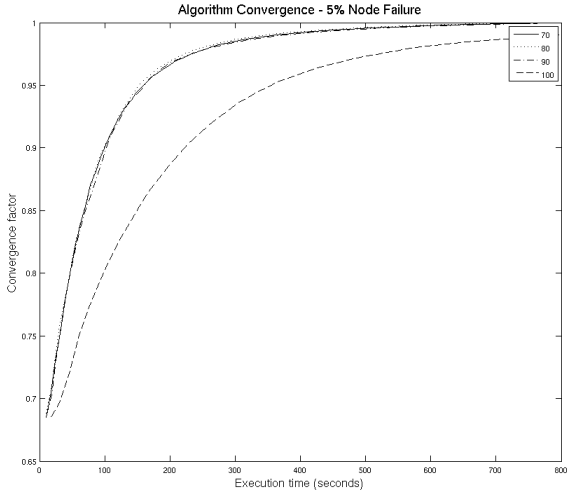

Fig. 5. Algorithm convergence for low degrees of synchronous behaviour - 70%, 80%, 90% and 100% (fully synchronous) - as a function of algorithm execution time, with 5% failure rate.

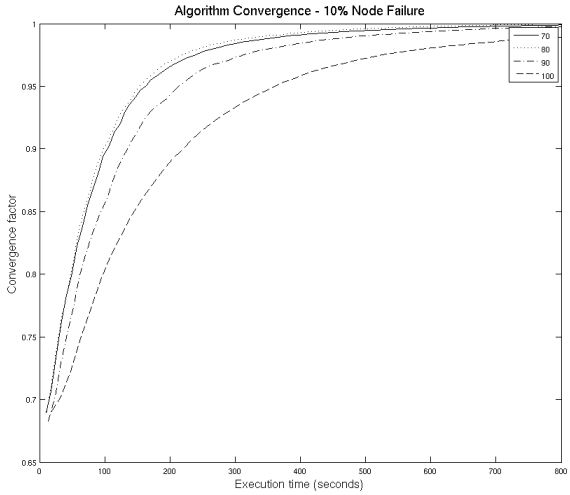

Fig. 6. Algorithm convergence for low degrees of synchronous behaviour - 70%, 80%, 90% and 100% (fully synchronous) - as a function of algorithm execution time, with 10% failure rate.

The performance losses shown in the PSPSO and the more synchronous PAPSO algorithms would not be evident on

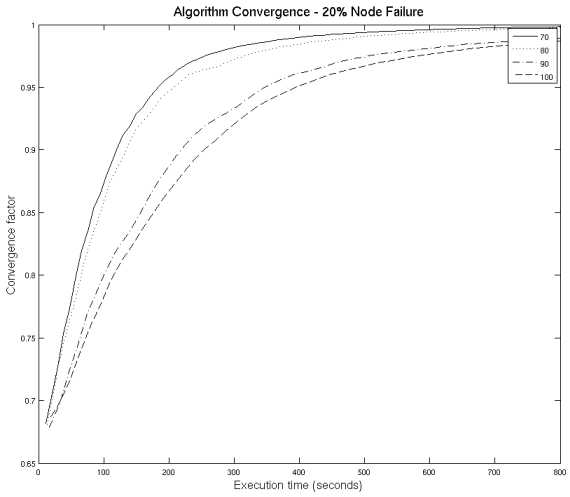

Fig. 7. Algorithm convergence for low degrees of synchronous behaviour - 70%, 80%, 90% and 100% (fully synchronous) - as a function of algorithm execution time, with 20% failure rate.

graphs displaying algorithm convergence as a function of completed algorithm iterations, highlighting the significance of examining the execution time of such parallel particle swarm algorithms in heterogeneous, fault-prone environments.

V. Conclusions

From the results presented in this paper it can be concluded that asynchronous particle updates, while negatively impacting algorithm convergence with respect to iterations completed, allow parallel distributed particle swarm algorithms to gracefully and efficiently handle node and solver heterogeneity as well as node, network and solver failure. Using asynchronous particle updates has been shown to increase parallel efficiency [1], which in turn decreases the computation time required for each algorithm iteration. This decrease in iteration time mitigates the decrease in algorithm convergence as a function of iterations completed which is introduced by the asynchronous update mechanism.

Simulations performed have shown that the more likely failures are, the more asynchronous the parallel particle swarm algorithm should be made. In fairly reliable environments such as fixed, dedicated high-performance computer clusters, mostly synchronous algorithms can be used, however if deploying into more volatile environments, such as distributed computer grids or non-dedicated peer-to-peer systems, the algorithms used should be made more asynchronous to counter the higher number of failures that can be expected.

Future work will be focused on investigating the effect of node heterogeneity and solver execution time on the PSPSO and PAPSO algorithms. It is expected that asynchronous updates will become even more advantageous the more heterogeneous the environment becomes, and the more variable the solver execution time becomes.

References

[1] B. Koh, A. D. George, R. T. Haftka and B. J. Fregly, "Parallel asynchronous particle swarm optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578-595, 2006.

[2] G. Venter and J. Sobieszczanski, "Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization," *Structural and Multidisciplinary Optimization*, vol. 26, no. 1-2, pp. 121-131, 2004.

[3] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka and A. D. George, "Parallel global optimization with particle swarm algorithm," *International Journal for Numerical Methods in Engineering*, vol. 61, pp. 2296-2315, 2004.

[4] G. Venter and J. Sobieszczanski-Sobieski, "A Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations," *Journal of Aerospace Computing, Information, and Communication*, vol. 3, no. 3, pp. 123-137, 2006.

[5] A. Carlisle, G. Dozier, "An Off-The-Shelf PSO", *Proceedings of the 2001 Workshop on Particle Swarm Optimization*, 2001, pp. 1-6.

[6] S. Mostaghim, J. Branke and H. Schmeck, "Multi-Objective Particle Swarm Optimization on Computer Grids," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2007, pp. 869-874.

[7] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942-1948.

[8] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: methods and applications," Thesis (doctoral), University of Zurich, Zurich, Switzerland, 1999.