Sheffield Hallam University

Optimisation of maintenance scheduling strategies on the grid

SHENFIELD, Alex <http://orcid.org/0000-0002-2931-8077>, FLEMING, Peter, KADIRKAMANATHAN, Visakan and ALLAN, Jeff

Available from Sheffield Hallam University Research Archive (SHURA) at:

http://shura.shu.ac.uk/8310/

This document is the author deposited version. You are advised to consult the publisher's version if you wish to cite from it.

Published version

SHENFIELD, Alex, FLEMING, Peter, KADIRKAMANATHAN, Visakan and ALLAN, Jeff (2010). Optimisation of maintenance scheduling strategies on the grid. Annals of Operations Research, 180 (1), 213-231.

Copyright and re-use policy

See http://shura.shu.ac.uk/information.html

Sheffield Hallam University Research Archive
http://shura.shu.ac.uk

Annals of Operations Research manuscript No.
(will be inserted by the editor)

Optimisation of Maintenance Scheduling Strategies on the Grid

Alex Shenfield · Peter J. Fleming · Visakan Kadirkamanathan · Jeff Allan

Abstract The emerging paradigm of Grid Computing provides a powerful platform for the optimisation of complex computer models, such as those used to simulate real-world logistics and supply chain operations. This paper introduces a Grid-based optimisation framework that provides a powerful tool for the optimisation of such computationally intensive objective functions. This framework is then used in the optimisation of maintenance scheduling strategies for fleets of aero-engines, a computationally intensive problem with a high-degree of stochastic noise, achieving substantial improvements in the execution time of the algorithm.

Keywords Maintenance Scheduling, Evolutionary Optimisation, Grid Computing

Introduction

A fundamental shift in emphasis within the aero-engine manufacturing industry is leading to the adoption of power-by-the-hour contracts, where airlines make regular fixed payments to the engine manufacturers based on the hours flown by an engine and, in return, the manufacturers of the engine retain the responsibility for servicing and maintenance. As a result of this, the accurate prediction of support costs over the life-cycle of an engine is of the utmost importance. However, aero-engines operate in a highly complex and unpredictable environment, and as such it is impossible to produce a deterministic model for these support costs. Instead, stochastic simulations can be performed to provide cost estimates. It is also important for the engine manufacturers to devise maintenance scheduling strategies to minimise support costs and thus enable more competitive pricing of these contracts.

Soft Computing techniques such as Neural Networks, Fuzzy Logic, and Evolutionary Computation have been used to solve many complex real-world engineering problems. These techniques provide the engineer with a new set of tools that often outperform conventional methods in areas where the problem domain is noisy, stochastic

A. Shenfield · P. J. Fleming · V. Kadirkamanathan · J. Allan
Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, S1 3JD, UK
E-mail: a.shenfield@sheffield.ac.uk

or ill-defined. However, in the cases of Neural Networks and Evolutionary Computation especially, these tools can be computationally intensive.

Grid Computing offers a solution to the computationally intensive nature of these techniques. The Grid Computing paradigm is an emerging field of computer science that aims to offer "a seamless, integrated computational and collaborative environment" (Baker et al, 2002). Ian Foster defines a computational Grid as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" (Foster and Kesselman, 1999). Grid Computing is differentiated from conventional distributed computing by its emphasis on co-ordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations (Foster et al, 2001). These resources include software packages, compute resources, sensor arrays, data and many others.

The purpose of this paper is to introduce a Grid-enabled framework for optimisation of maintenance schedules. This framework will then be used to assist decision makers in planning maintenance scheduling strategies for aero-engines. This problem presents many challenges due to to its highly stochastic nature.

Section 1 will introduce evolutionary algorithms and give a brief overview of their application to scheduling problems. The core concepts of Grid computing used in our optimisation framework will be introduced in Section 2, and, in Section 3, a brief summary of related work will be given. The MEAROS simulation package used by Rolls-Royce to model the operational life-cycle of engines will be introduced in Section 4, and, in Section 5, the implementation of our Grid-based optimisation framework will be described. Section 6 will demonstrate the application of our framework to the planning of maintenance schedules for aero-engines, whilst Section 7 will discuss the results obtained using our framework and present some conclusions and ideas for further work.

1 An Introduction to Evolutionary Algorithms

1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are an optimisation technique utilising some of the mechanisms of natural selection (Goldberg, 1989). EAs are an iterative, population based method of optimisation that are capable of both exploring the solution space of the problem and exploiting previous generations of solutions. Exploitation of the previous generation of solutions is performed by a selection operator. This operator gives preference to those solutions which have high fitness when creating the next generation of solutions to be evaluated. Exploration of the solution space is performed by a mutation operator and a recombination operator and helps to ensure the robustness of the algorithm by preventing the algorithm from getting stuck in local optima.

Evolutionary Algorithms evaluate candidate solutions based on pay-off information from the objective function, rather than derivative information or auxiliary knowledge. This ensures that EAs are applicable to many different problem domains, including those where conventional optimisation techniques (such as hill-climbing) may fail. Evolutionary Algorithms are also robust in the presence of noise due to their population based nature. Because EAs maintain a population of candidate solutions, each generation contains more information about the shape of the fitness landscape than would

be available to conventional, non-population based optimisation methods (Michalewicz and Fogel, 2000).

Evolutionary Algorithms have been used to solve problems across many different disciplines. EAs have been used in such diverse fields as Economics and Social Theory (Axelrod, 1987), Robotics (Pratihar et al, 1999) and Art (Sims, 1991). For many non-trivial real-world applications the evaluation of the objective function is performed by computer simulation of the system. For example, in the optimisation of controller parameters for gas turbine aero-engines (Fleming et al, 2002), a computer model of the engine is used to calculate the values of the objective functions for a given controller design.

The use of computer simulations to evaluate the objective function leads to some new issues. To ensure that the results gained from the evolutionary algorithm are meaningful, the simulation must be complex enough to capture all the relevant dynamics of the true system. However, assuming that this level of complexity is obtainable, the simulation may be very computationally expensive. As EAs are population based methods, the simulation must be run many times. In a typical evolutionary algorithm this could involve running the simulation 10,000 times.

1.2 Scheduling Applications of Evolutionary Algorithms

Finding good solutions to industrial scheduling problems is of great importance, since both production rates and plant costs are dependent on work schedules. Evolutionary algorithms have had some success in solving the canonical Job-Shop Scheduling Problem (Davis, 1985; Mesghouni et al, 2004), a problem that is representative of industrial tasks ranging from assembling cars, to scheduling aircraft maintenance. Recent focus in the EC community has been on generating robust and flexible job shop schedules (Jensen, 2003). Other scheduling problems solved by EAs include planning maintenance for the (UK) national grid (Langdon, 1995) and university course timetabling (Lewis and Paechter, 2005).

1.3 Parallel Evolutionary Algorithms

The computationally expensive nature of the evolutionary algorithm evaluation process has motivated the development of parallel EAs. Early approaches to the implementation of parallel evolutionary algorithms can be classified into two categories which still apply today: single-population, globally parallel EA implementations and EA implementations with multiple communicating populations (Cantú-Paz and Goldberg, 1999).

Single-population parallel evolutionary algorithms consist of a single *panmictic*[1] population maintained globally. This form of parallelism may be effectively exploited using the well established Master-Worker paradigm from parallel computing (see Figure 1). Typically the evaluation of candidate solutions in the algorithm is distributed amongst the worker nodes whilst the master node applies the evolutionary operators, such as selection and variation, centrally to the whole population (Fogarty and Huang, 1991). Chipperfield and Fleming (1995) also describe a similar scheme where both the

[1] A panmictic population is one where all individuals are potential partners, i.e. there are no geographical restrictions to the mating of pairs of individuals.

evaluation of candidate solutions and the variation operators are performed by the worker nodes.

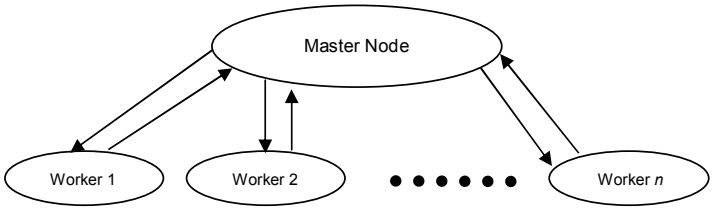

Fig. 1: The Master-Worker Paradigm

These single-population, globally parallel EAs represent an important case of parallelism because they are functionally equivalent to serial EAs. This means that existing EA theory and design guidelines can easily be applied to their use (Cantú-Paz and Goldberg, 1999). Although this type of strategy does not exploit all the parallelism inherent in the evolutionary algorithm, substantial improvements in performance can be achieved - especially in cases where the evaluation of candidate solutions is significantly more computationally expensive that the evolutionary operators themselves (Chipperfield and Fleming, 1995).

Evolutionary algorithms with multiple communicating populations can be further divided into those that implement a *coarse-grained* parallelism and those that implement a *fine-grained* parallelism. Algorithms that implement a coarse-grained parallelism (also known as *island* or *migration* EAs) introduce a degree of geographical isolation into the search. The population is divided up into multiple subpopulations (known as *demes*), with each subpopulation evolving independently (Chipperfield and Fleming, 1995). Periodically migration occurs to allow an exchange of information between subpopulations (Rivera, 2001). Figure 2 shows an example of a coarse-grained island EA using a ring topology, although it should be noted that other topologies and interconnections are equally applicable.

Fine-grained parallel EAs (also known as *diffusion* EAs) treat the population as a single continuous structure (Chipperfield and Fleming, 1995). In these diffusion EAs a grid is formed to cover the population surface, and each member of the population is assigned to a node in that grid (with each node ideally hosted on a separate processor). The evolutionary operators are then applied to individuals in the same local neighbourhood (usually chosen to be the adjacent nodes). Rivera (2001) notes that the topology of the network in diffusion EAs strongly determines the behaviour of the algorithm.

The decision as to which of these forms of parallelisation to implement must consider several factors, such as ease of implementation and use, and the potential performance gains from parallelisation. Single-population parallel EAs are often easiest to implement and use, since experience gained with sequential EAs is directly applicable. In contrast, the implementation of parallel EAs with multiple communicating populations requires the consideration of extra design choices. For instance, the use of an island model EA requires the algorithm designer to choose the number of demes, the

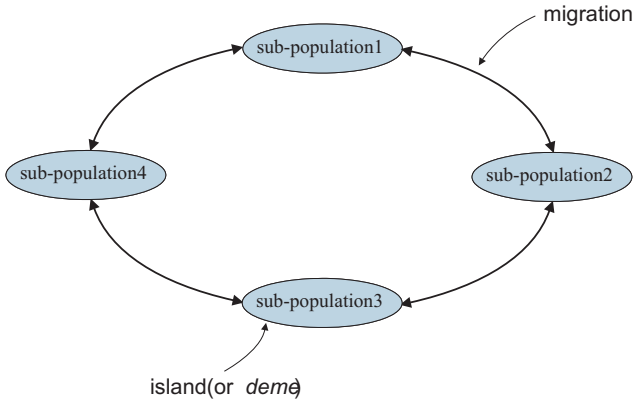

Fig. 2: A Coarse-Grained Island Evolutionary Algorithm in a Ring Topology

population topology, and the mutation rate, as well as choosing values for the standard evolutionary parameters. This increases the complexity of the parallel EA since each of these parameters influences the efficiency of the algorithm and the quality of the overall solution. Whilst some authors have reported improved convergence using EAs with multiple communicating populations (Grosso, 1985; Tanese, 1987; Starkweather et al, 1991), it should be noted that this is heavily dependent on the values chosen for the extra parameters.

2 Grid Computing Technologies

The concept of Grid computing is not new. As far back as 1969 Len Kleinrock suggested:

> "We will probably see the spread of 'computer utilities', which, like present electric and telephone utilities, will serve individual homes and offices across the country." (Klienrock, 1969)

However, it is only recently that technologies such as the Globus Toolkit (Foster and Kesselman, 1999) have emerged to enable this concept to be achieved. The Globus Toolkit is an open-source, community-based set of software tools to enable the aggregation of compute, data, and other resources to form computational grids. Since version 3, the Globus Toolkit has been based on the Open Grid Services Architecture (OGSA) introduced by the Globus Project. OGSA builds on current Web Service concepts and technologies to support the creation, maintenance, and application of ensembles of services maintained by virtual organisations (Foster et al, 2002).

2.1 Web Services

A Web Service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the

Web service in a manner prescribed by its description using SOAP messages" (W3C Working Group, 2004). Web Services are accessible through standards-based internet protocols such as HTTP and are enabled by three core technologies (Chappell and Jewell, 2002):

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)

These technologies work together in an application as shown in Figure 3. The Web Service client queries a UDDI registry for the desired service. This can be done by service name, service category, or other identifier. Once this service has been located the client queries the WSDL document to find out how to interact with the service. The communication between client and service is then carried out by sending and receiving SOAP messages that conform to the XML schema found in the WSDL document.

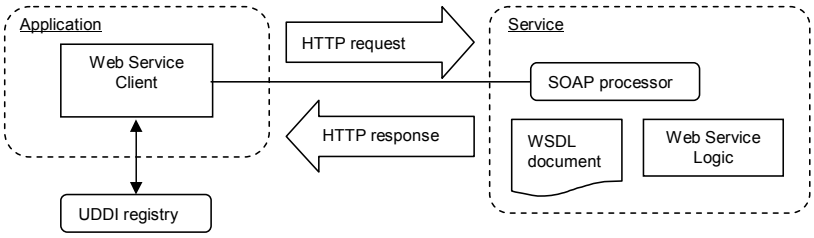

Fig. 3: Interaction between Web Service Technologies

2.2 Open Grid Services Architecture

The Open Grid Services Architecture forms the basis for the Globus Toolkit. OGSA represents computational resources, data resources, programs, networks and databases as services. These services utilise the Web Services technologies mentioned in Section 2.1. There are three main advantages to representing these resources as services:

1. *It aids interoperability.* A service-oriented view addresses the need for standard service definition mechanisms, local/remote transparency, adaptation to local OS services, and uniform semantics (Foster et al, 2002).
2. *It simplifies virtualisation.* Virtualisation allows for consistent resource access across multiple heterogeneous platforms by using a common interface to hide multiple implementations (Foster et al, 2002).
3. *It enables incremental implementation of Grid functionality.* The provision of Grid functionality via services means that the application developer is free to pick and choose the services that provide the desired behaviour to their application.

3 Related Work

Tan et al (2003) and Fung et al (2004) have both developed parallel evolutionary computing environments in Java to solve single-objective optimisation problems. However, neither of these environments is well suited for use in a large-scale, multi-site computational Grid. The distributed evolutionary computing system proposed by Tan et al (2003) is based on the island model of parallel EAs (see Section 1.3) and uses a small number of peers to host multiple communicating populations. Whilst in theory the island model should scale well to larger numbers of peers, Fernandez et al (2003) have shown that this scalability is difficult to exploit in practice. Fung et al (2004) propose a Java-based parallel platform for evolutionary computation using a Distributed Shared Memory (DSM) architecture. This architecture is unsuitable for use in a large-scale, multi-site computational Grid due to its tightly coupled nature.

Tanimura et al (2002) have proposed a middleware system for enabling evolutionary optimisation in a Grid computing environment. This system requires the application designer to develop suitable evolutionary operators and implement them according to a common set of interfaces. Tanimura et al (2002) use this middleware system to solve a single-objective optimisation problem by constructing a parallel simulated annealing algorithm. Abdalhaq et al (2002) also use the concept of Grid computing to perform single-objective optimisation using evolutionary computation by developing a *Black Box Optimisation Framework (BBOF)* in C++ to optimise a computer simulation of a single-objective forest fire propagation problem. This optimisation process is run on a single compute cluster managed by the Condor resource management system. Herrera et al (2005) have also implemented a Grid-Oriented Genetic Algorithm (GOGA). This GOGA is based on the fully connected island model of parallel evolutionary computation and, as such, suffers from the problems outlined in Section 5.1. Herrera et al's (2005) grid-oriented genetic algorithm uses the GridWay (Distributed Systems Architecture Group, Universidad Complutense de Madrid, 2007) meta-scheduling framework to distribute and manage subpopulations in a small Grid testbed made up of 4 machines.

Xue et al (2004) and Song et al (2004) have implemented a single-objective genetic algorithm in a service oriented architecture to solve a 2D aerodynamic design optimisation problem. Their approach is similar to that taken in this paper; however, the distributed evaluation of candidate solutions in Song et al (2004) is performed using a single compute cluster located at a single site, whereas the evaluation of candidate solutions described in this paper uses computational resources located at multiple geographically distributed sites. Another SOA approach to the implementation of parallel evolutionary algorithms is that taken by Lim et al (2007). Lim et al (2007) have implemented a hierarchical parallel evolutionary algorithm in a distributed computational Grid, with multiple subpopulations distributed across the Grid resources. The evaluation of candidate solutions in these subpopulations is then performed using the Master-Worker paradigm previously illustrated in Figure 1. This approach suffers from the potential problems outlined in Section 5.1 later, and, although results are presented for the execution times of the EA, not much information is given about the quality of the final solutions produced by the optimiser.

4 Life-Cycle Simulation of Aero-Engines

The Modular Engine Arisings, Repair and Overhaul Simulation (MEAROS) package was developed to enable Rolls-Royce and the Ministry of Defence to evaluate the operation, maintenance and supply of aircraft engines (Rolls-Royce, 2002). Although designed for the aero-engine manufacturing industry, the simulation can equally be applied to ships, land vehicles and power generation (Argyle, 2006). The modelling capability of the MEAROS software is extensive and can be used to model the operation of fleets of engines with an arbitrary number of modules (Rolls-Royce, 2002). Theoretically there is no limit to the size of fleets that can be modelled by the software; however, in practice this is limited by the computational effort needed to model large numbers of engines.

Results produced by the simulation contain a lot of stochastic noise due to the probabilistic models used to simulate component failures. As such, the simulation has to be run multiple times and averaged to reduce the effect of this noise. Figure 4 shows that the standard deviation of the aggregate maintenance cost reduces with the number of runs of the model. It can also be seen from Figure 4 that the improvement in the standard deviation tails off substantially after 100 passes. In practice this means that the benefit from running more than 100 passes of the model is outweighed by the additional computational cost.

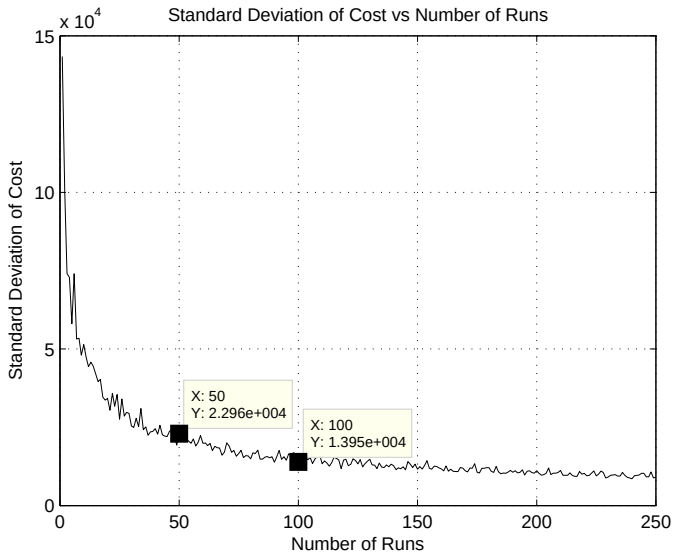

Fig. 4: Plot of the Standard Deviation of Aggregate Maintenance Cost Against Number of Runs of the Model

Originally MEAROS was used for predicting the number of spares needed to maintain a set level of operational availability. However, many of the parameters in the model are customisable (such as the Weibull slope parameters of the failure distributions of engine modules, the stock levels, and the maintenance scheduling strategies

used) and can therefore be optimised with respect to some objective (for instance the support costs or the operational availability).

Maintenance in the simulation is performed after an arising occurs. The main causes of arisings are either the expiry of a *hard-life*[2] or an in-service failure such as foreign object damage (Rolls-Royce, 2002). Once an arising occurs, the engine must be removed from the aircraft wing and the module that caused the arising must be reconditioned or replaced. However, as the removal of the engine from the wing is typically the most expensive part of a maintenance shop visit, this provides the ground crew with the chance to perform opportunistic maintenance on the other modules in the engine. If one of the other modules in the engine has exceeded its *soft-life*[3] then it should also be reconditioned or replaced whilst the engine is removed from the wing.

5 A Grid-Based Framework for Optimisation Using Evolutionary Algorithms

5.1 Parallelisation of the Evolutionary Algorithm

In Section 1.3 two categories of possible parallelisation strategies for evolutionary algorithms were described: single-population, globally parallel EAs and EAs with multiple communicating populations. A major drawback with EAs that use multiple communicating populations is the difficulty in setting the extra parameters needed. Whilst guidelines exist in the literature for choosing some of these parameters (such as the migration rate and migration interval), no general guidelines were found for the optimal number of subpopulations to use. In fact results presented in Shenfield (2007) suggest that the optimal number of subpopulations varies from problem to problem, and thus no general guidelines can be given.

The number of subpopulations has been shown to be a key factor in the quality of the final solutions produced by a parallel evolutionary algorithm (Shenfield, 2007). However, in a Grid computing environment the number of subpopulations and population structure may be determined by the configuration of the Grid resources and may therefore not be optimal for the problem under consideration. For this reason it was decided to use the single-population master-slave implementation, since it does not require the choice of these extra parameters. The single-population parallel model also allows experience from implementing sequential EAs to be easily applied.

5.2 Implementation in a Service-Oriented Architecture

We have chosen to implement our Grid-enabled framework for optimisation using evolutionary algorithms in a Service-Oriented Architecture (SOA). The service-oriented architecture approach to Grid computing is well suited to the kind of master-worker parallelism chosen for our optimisation framework since the client can act as the master node (by generating and varying the population), and the service can act as the worker (by evaluating the individual candidate solutions). A key advantage to providing the

[2] Hard-lives are usually assigned to safety critical components and represent the age at which that component must be replaced.

[3] Soft-lives represent the age whereby a component should be replaced at the next opportunity.

components of our optimisation framework as services is that the functionality can be accessed via the HTTP protocol, thus allowing the services to be easily integrated into an Internet portal and accessed by any device with a web browser (such as a PDA).

In the implementation of our optimisation framework (see Figure 5) there are two different types of service. One service type exposes the operations of the evolutionary algorithm to the client, and the other provides the ability to run evaluations of the objective function on the resources of a computational Grid. These services interact (as shown in the pseudo-code listed in Figure 6) to provide a flexible grid-enabled optimisation framework.

These services are written in Java and deployed using the open source Apache Tomcat/Apache Axis web service development platform. Apache Tomcat provides a robust, cross-platform web application container to host the web service, whilst Apache Axis provides a SOAP (Simple Object Access Protocol) engine that enables web services and web service clients to process SOAP messages sent across a network. Whilst these tools require a degree of customisation for the specific environment they are used in, they do greatly simplify the development of both web services and web service clients. For example, Apache Axis provides a library of utility functions to enable web service clients to connect to web services by simply providing the service location (i.e. the URL), and also to automatically convert Java (or C++) RPCs (Remote Procedure Calls) to SOAP messages to enable the client to interact with the target service.

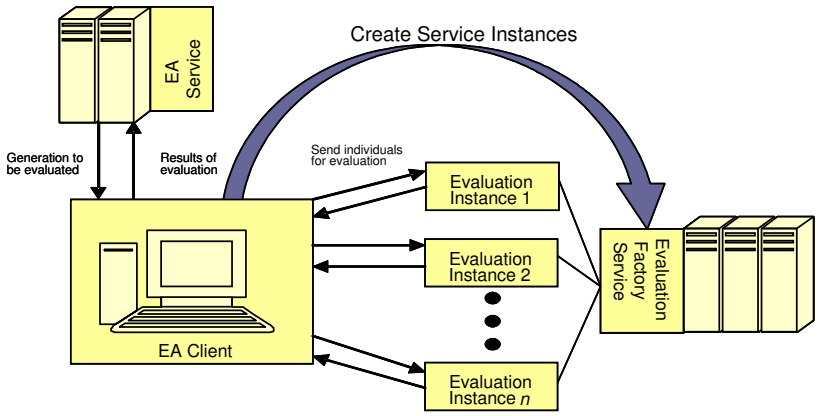

Fig. 5: The Implementation of the Optimisation Framework

The Evolutionary Algorithm Web Service

For the results presented in Section 6.1, our optimisation framework was used in a Genetic Algorithm architecture with real valued representations of the decision variables. Fogel and Ghoziel (1997) have shown that there is no intrinsic advantage in choosing one bijective representation over another, although particular representations may be

```
PROCEDURE GridEnabledEA:

    // connect to the evolutionary algorithm web service and
    // the evalution grid service
    EAService = connectEAService();
    EvalService = connectEvaluationService();

    // locate Grid resources
    resources = EvalService.findResources();

    // initialise the evolutionary algorithm
    solutions = EAService.initialise();

    // evaluate the initial generation in parallel
    EvalService.distributeSolutions(resources, solutions);
    EvalService.evaluateSolutions();

    // run the evolutionary algorithm until some termination
    // criterion is met
    WHILE not finished DO:

        // apply evolutionary operators
        EAService.selection(solutions);
        EAService.recombination(solutions);
        EAService.mutation(solutions);

        // evaluate the current generation in parallel
        EvalService.distributeSolutions(resources, solutions);
        EvalService.evaluateSolutions();

    END

END
```

Fig. 6: Pseudo-code Describing the Interaction Between Services

more computationally tractable or efficient for certain problems. Consequently modern EA practice emphasises choosing a representation that is appropriate for the problem under consideration. As the decision variables in the problem considered in this paper are continuous it is intuitive to use a real-valued representation (Michalewicz and Fogel, 2000).

Selection in our algorithm was performed using Stochastic Universal Sampling (Baker, 1987) which guarantees sampling with zero bias and minimum spread, and is generally considered superior to other selection schemes for many problems (Hancock, 1994). The extended intermediate recombination operator and BGA mutation operator from (Mühlenbein and Schlierkamp-Voosen, 1993) were used to introduce variation into the population and prevent the evolutionary process from stagnating.

It is important to note that adding additional functionality (such as alternative evolutionary operators) to an optimisation routine using our framework can easily be accomplished simply by implementing additional services.

The Evaluation Grid Service

The distribution and management of computational tasks across a diverse set of distributed heterogeneous resources is a key issue in Grid computing. Many resource management systems exist to address the problem of scheduling at a local level, but the dynamic and decentralised nature of computational Grids provides additional challenges not addressed by these systems.

The evaluation grid service shown in Figure 5 provides the ability to evaluate multiple candidate solutions in parallel using the resources of the White Rose Grid (see Section 5.3). To do this it uses an *application-centric* meta-scheduler[4] to distribute the objective function evaluations across the available resources with the aim of minimising the mean response time of jobs through the system (i.e. maximising the throughput of objective function evaluations). This application-centric approach is similar to that taken by the AppLeS project (Berman et al, 1996). However, a key difference is in the use of response time information from previously completed generations to provide estimates of the computational capacity of the available resources, rather than explicitly querying the computational resources for their status (a process that can be both complex and time intensive). Results presented in Shenfield (2007) have shown that this approach performs extremely well in complex distributed and dynamic environments (such as computational Grids).

Our evaluation grid service exposes three methods to the Grid-enabled optimisation client (as shown in the pseudo-code in Figure 6):

1. **findResources()** - this method queries a database to discover what Grid resources are available and obtain some initial information about their states.
2. **distributeSolutions(resources, solutions)** - this method uses the application-centric meta-scheduler outlined above to calculate the optimal workload allocation (i.e. the optimal number of candidate solutions to send to each resource) with respect to the mean response time of jobs through the system, for a given set of resources. This optimal workload allocation is calculated using elements of queueing theory (see Kleinrock (1975) for more details) to minimise the mean response time for the evaluation of candidate solutions. It then transfers these candidate solutions to the Grid resources using either SFTP (the Secure File Transfer Protocol) or GridFTP[5].
3. **evaluateSolutions()** - this method starts a job manager daemon on the Grid resources to manage the objective function evaluations. It does this by using the local resource management system (in the case of the White Rose Grid resources this would be Sun Grid Engine) to run as many instances of the evaluation function as there are candidate solutions. These evaluation function instances are then queued by the local scheduler and run when appropriate compute resources become available. The results are then returned to the client.

[4] Meta-scheduling is an approach where jobs are submitted via local resource management systems rather than directly to the actual machines themselves.

[5] GridFTP offers potential performance benefits when dealing with large data-sets, but requires the administrators of the Grid resources to provide a GridFTP server.

5.3 The White Rose Grid

The White Rose Grid (The White Rose University Consortium, 2007) is a multi-institutional computational Grid launched in 2002 by the universities of Sheffield, York and Leeds. The main objective of the White Rose Grid project is to support e-Research by providing users with access to large amounts of heterogeneous compute resources. The White Rose Grid currently consists of five high-performance compute nodes located at three different sites (see Figure 7 for an overview of the network topology), and in 2003 was awarded the status of e-Science Centre of Excellence.

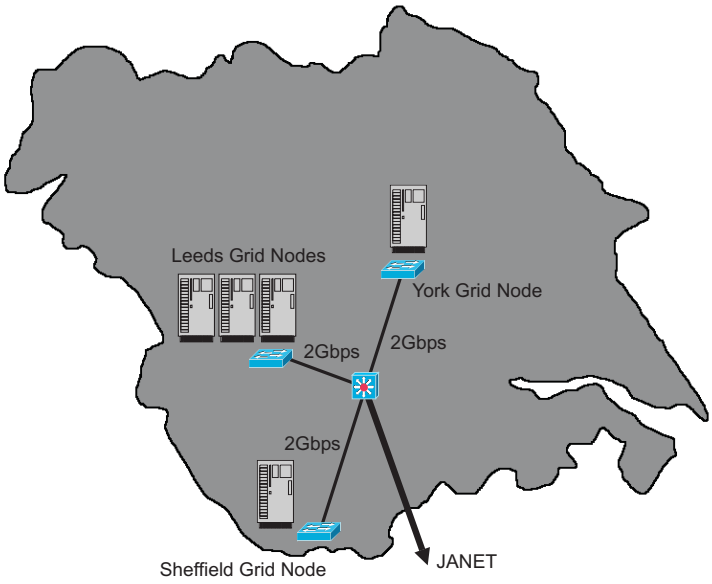

Fig. 7: An Overview of the Network Topology of the White Rose Grid

The three participating institutions in the White Rose Grid consortium reserve 75% of their Grid resources for users within their institution and allocate the remaining 25% to other users of the Grid. The current Grid resources available are outlined in Table 1. These resources are connected by the high bandwidth Yorkshire and Humberside Metropolitan Area Network (YHMAN), and are managed at a local level by Sun Grid Engine. However, there is currently no production quality grid-level meta-scheduler available to enable a scientist or engineer to transparently utilise all these multi-institutional resources.

6 Maintenance Scheduling Strategy Optimisation

We have used our Grid-based optimisation framework to optimise the aero-engine maintenance scheduling strategy across a fleet of aircraft with the aim of minimising the

Table 1: White Rose Grid Resources

Sheffield	
Iceberg	Iceberg is a compute cluster consisting of 320 2.4GHz AMD Opteron processors running the Scientific Linux operating system. 160 of these processors are available for general use, whilst the other 160 are reserved for the GridPP project (The GridPP Project, 2007).
York	
Pascali	Pascali consists of a large memory cluster for jobs with big memory requirements and a Beowulf cluster with 24 nodes. The Beowulf cluster is based on dual core AMD Opteron processors, with each node comprising of two processors. These clusters run the Scientific Linux operating system.
Leeds	
Maxima	Maxima is a constellation of shared memory SMP systems based on Sun Fire 6800 and V880 servers. This node offers 60 UltraSPARC III processors running the Solaris operating system.
Snowdon	Snowdon is a cluster of 128 dual Intel Xeon processor based compute nodes. This Grid node offers a total of 256 processors for dedicated batch use and runs the Linux operating system.
Everest	Everest is a cluster based on AMD Opteron dual core processors running the Linux operating system. It consists of 66 dual processor Sun V20z servers and 8 quad processor Sun V40z servers.

total maintenance cost. The maintenance scheduling strategy consists of a set of soft-lives for the modules in the engine which determine when opportunistic maintenance is performed (see Section 4 for more details). Crocker and Kumar (2000) have shown that, for relatively small engine module costs, there is likely to be an optimum value of soft-life which minimises the total maintenance cost of an engine. This is because soft-lives that are too low result in engine modules being reconditioned or replaced during every maintenance shop visit; whilst soft-lives that are too high lead to cheaper, but more frequent, shop visits[6] (since high soft-lives result in very little opportunistic maintenance being performed).

The maintenance scheduling strategy was chosen for optimisation because it is one of the few parameters affecting support costs that is easily modifiable once the engine has gone into service. It is inexpensive to vary when compared to post-production design changes and can be quickly implemented across an engine range (Argyle and Tubby, 2002).

Our optimiser used a floating point representation for each of the five decision variables (i.e. engine module soft-lives) that made up a single candidate solution. The lower and upper bounds on these decision variables were 0 hours and 5000 hours[7] respectively. The evaluation of candidate solutions in our optimiser was performed using the MEAROS engine life-cycle model described in Section 4, in conjunction with the simple cost model shown in Table 2. This cost model was developed in partnership with Rolls-Royce and represents a hypothetical five module aero-engine. The costs given are for the removal and reconditioning of the modules in the engine, whilst the

[6] And, as noted in Section 4 the removal of the engine from the wing is typically the most expensive part of a maintenance shop visit.

[7] This upper limit of 5000 hours was chosen as it accounts for over 99% of failures for the engine module with the longest life.

scale and slope parameters of the Weibull distribution represent the characteristic life of a module and the failure distribution of a module respectively.

Table 2: Cost Model and Weibull Failure Distribution Parameters

	Cost	**Scale**	**Slope**
Engine	3000	N/A	N/A
Module 1	200	1000	1
Module 2	1000	800	2.5
Module 3	900	700	3
Module 4	800	2000	2
Module 5	1200	1500	1.5

The MEAROS model was used to simulate the effect of a given set of soft-lives on a fleet of 25 engines over a 10 year period, and, as mentioned in Section 4, the results were averaged over 100 passes of the model so as to reduce the effects of stochastic noise in the simulation. Evaluation of a single candidate solution using the above configuration took in the order of 1.5 seconds on a Intel Pentium 4 based PC running at 3.0GHz.

The evolutionary algorithm web service described in Section 5.2 was used to initialise the population and to perform selection and variation (with a recombination rate of 0.8 and a mutation rate of 0.1) on each generation, and the evaluation grid service was used to distributed the evaluation of candidate solutions amongst the resources of the White Rose Grid (see Section 5.3). As many computational resources at the universities of Sheffield, Leeds and York were used during the optimisation process as the site policies and local schedulers would allow, with the application-centric meta-scheduler described briefly in Section 5.2 (and in more detail in Shenfield (2007)) used to maximise the throughput of objective function evaluations through the available resources.

6.1 Optimisation Results

The EA was run multiple times with a population size of 50 individuals. However, as similar results were obtained from each execution of the algorithm, the results presented in Figure 8 and Figure 9 are from a single representative run of the EA only. Figure 8 shows that the mean value of the population (the solid line in the figure) exhibits convergence after around 15-20 generations. It can also be seen from Figure 8 that the diversity of the population (each individual in the population is shown by a dot in the Figure) is substantially reduced as the search progresses. Using one-at-a-time (OAT) sensitivity analysis[8] (Hamby, 1994), it is possible to show that the final solution produced by the EA is optimal - since varying the final set of decision variables (i.e. the set of soft-lives that make up the maintenance scheduling strategy) produced by the optimiser does not yield a better solution.

[8] One-at-a-time sensitivity analysis involves varying each of the soft-lives that make up the best solution produced by the optimiser one after the other (whilst keeping the other soft-lives constant at the value found by the optimiser) and observing the influence of the changes on the model output.

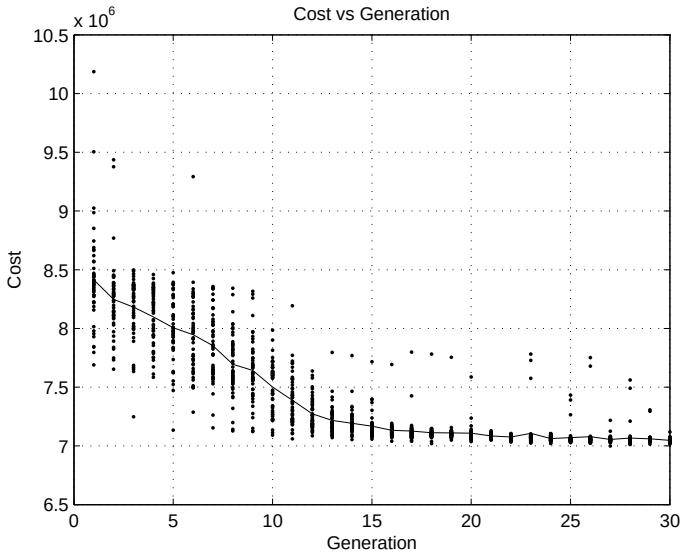

Fig. 8: Plot of Cost of Maintenance Scheduling Strategy Against Number of Generations

Figure 9 shows the distributions of the engine module soft-lives from the 30 generations after the optimiser has converged with the quartile values marked on the plots by vertical lines. It can be seen from Figure 9 that both the soft-lives found by the optimiser and the inter-quartile ranges of their distributions are relatively high for modules 1 and 5. This is because the failure distributions for these modules indicate that they fail randomly[9] and nearly randomly, respectively. Assigning soft-lives to these components will therefore just increase the overall maintenance cost. Modules 2, 3, and 4 are all assigned lower soft-lives by the optimiser, indicating that preventative maintenance of these components can reduce the overall cost. Modules 2 and 3 should be replaced often, since they have relatively short characteristic lives, whilst module 4 has a much longer characteristic life and therefore should not be replaced as much. Analysis of the model output indicates that module 4 rarely causes arisings.

Table 3 shows the average execution times from the optimisation of the aero-engine maintenance scheduling problem described in this paper. These results are averaged over 5 runs for both the single workstation results and the results obtained using our Grid-based framework for optimisation using evolutionary algorithms. The execution times from our Grid-based optimisation framework were taken at different times of the day, so as to reduce the effect of system load[10] on the results presented in Table 3. The best execution times obtained by our Grid-based optimisation framework were 521 seconds to evaluate 30 generations, and 897 seconds for 50 generations (these results were taken on a Sunday morning). As Table 3 shows, the use of our Grid-based optimisation framework can considerably reduced the time taken to optimise the aero-engine maintenance scheduling problem considered here.

[9] A failure distribution with a slope of one indicates a component will fail randomly.

[10] System load was typically lowest in the early morning and at weekends, and highest in the afternoon.

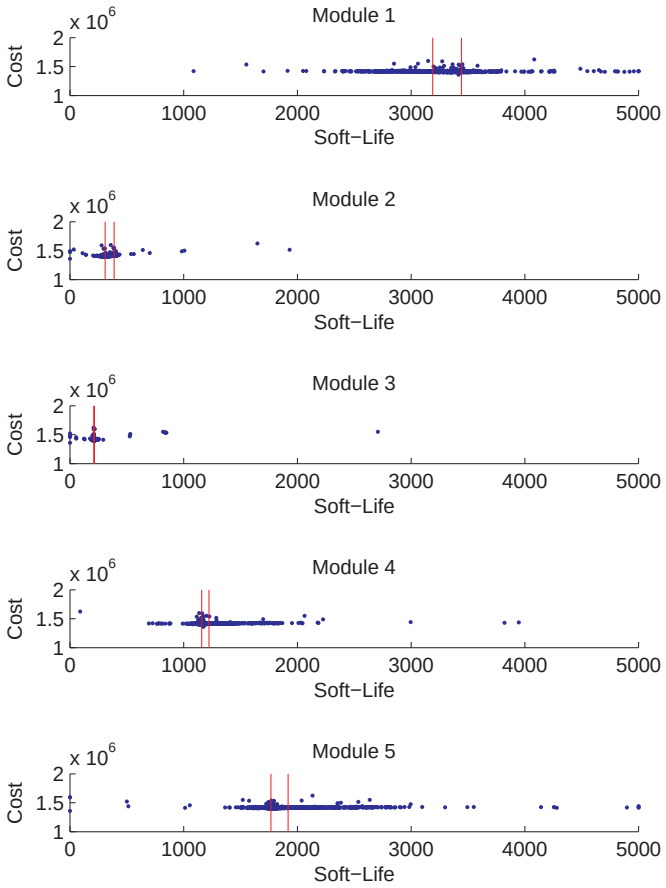

Fig. 9: Optimised Soft-Lives for a Five Module Aero-Engine

Table 3: Execution Times for the Optimisation of Maintenance Scheduling Strategies

Single Workstation		**Computational Grid**	
30 generations	50 generations	30 generations	50 generations
1981 seconds	3372 seconds	826 seconds	1354 seconds

7 Discussion, Conclusions and Further Work

The aero-engine model used for the maintenance scheduling strategy optimisation presented in this paper represents a simplified version of a real-world system (primarily due to computational constraints encountered in previous work (Argyle, 2006)). However, the speed up obtained using our Grid-based optimisation framework will enable the optimisation of larger scale models - such as those applied to bigger fleets of aircraft or those with higher fidelity models of engines (i.e. those comprising of a larger number of modules). This speed-up also enables a decision maker to run the optimisation

process multiple times with alternative cost models to get a clearer understanding of the problem space.

Whilst the implementation of the framework described in this paper has concentrated on the application of an evolutionary algorithm to a single objective maintenance problem, our framework is easily extensible to both multi-objective problems and to the implementation of alternative optimisation methods such as ant-colony optimisation or particle swarm optimisation. This extensibility is a result of implementing the components of our framework as services and is an important advantage of the Grid computing approach taken here. Grid computing is not just about increased computational speed, but also about providing transparent on-demand access to computational resources (ranging from computer processors to software) by taking a service-oriented view of application architectures. This approach also greatly simplifies the maintenance of our system, allowing upgrades to take place without impacting on the end user. Further work is planned to extend this optimisation framework to perform multi-objective optimisation of schedules for more complex logistics and supply chain operations.

The Grid-enabled optimisation framework proposed here is primarily suited to computationally expensive objective function evaluations, such as the one described in this paper. This is due to both the communication overheads inherent in evaluating candidate solutions in a distributed manner and the high overall utilisation of the available Grid resources. Figure 10 shows the effectiveness of our framework as the complexity of the objective function increases. In this experiment each generation consisted of 50 individuals, and the computational complexity of the objective function evaluation was varied. No evolutionary operators were applied between generations since the computational expense of these operators is minimal. Multiple runs of the experiment were performed, and the results averaged.

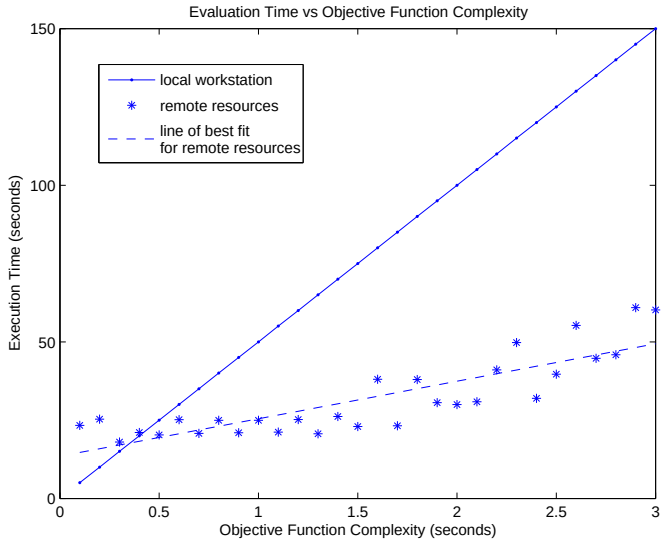

Fig. 10: A Plot of Evaluation Time Against Objective Function Complexity

As can be seen from Figure 10, using this Grid-enabled framework for optimisation of computationally trivial objective functions may result in a decrease in performance when compared to a sequential EA. However, for objective functions that take over 0.5 seconds to evaluate a single individual, substantial savings can be achieved in the total execution time of the algorithm. The results presented in this Section also show that the potential speed-up that can be achieved by a problem increases with the computational complexity of the objective function evaluations. This is because as the complexity of the problem increases, the amount of time spent waiting for the jobs to run becomes less significant and thus the potential speed-up increases. We can see from Figure 10 that the combined overheads of communication time and time spent waiting for the job to be run by the local scheduler are in the order of 20 seconds, although this may change as the load on the system varies.

It is expected that further research and development into Grid middleware, resource management systems, and network infrastructure will result in reductions in the communication overheads present in the proposed framework. Negotiating Service Level Agreements (SLAs) with the resource providers would also guarantee a minimum level of service for the objective function evaluations, enabling the proposed framework to provide increased performance for less computationally expensive problems. However, this framework is not intended to replace sequential EAs in cases where the performance of sequential EAs is satisfactory.

Acknowledgements This work and that of the DAME project are supported by the Grant Number GR/R67668/01 from the Engineering and Physical Research Council (EPSRC) in the U.K., and through contributions from engineers at Rolls-Royce and Data Systems and Solutions. The authors also gratefully acknowledge the support of the BROADEN project (part-funded by a collaborative R&D grant under the DTI Technology Programme), and the valuable suggestions of the anonymous referees.

References

Abdalhaq B, Cortes A, Margalef T, Luque E (2002) Evolutionary optimization techniques on computational grids. In: Proceedings of the International Conference on Computer Science (ICCS2002), Springer-Verlag, pp 513–522

Argyle JPM (2006) Optimisation of operational cost with application to an aerospace engine system. PhD thesis, University of Sheffield

Argyle JPM, Tubby J (2002) Integrated logistics support optimisation. Tech. Rep. RRUTC/Shef/R/02006, Rolls-Royce PLC

Axelrod R (1987) The evolution of strategies in the iterated prisoner's dilemma. In: Davis L (ed) Genetic Algorithms and Simulated Annealing, Morgan Kaufmann, pp 32–41

Baker JE (1987) Reducing bias and inefficiency in the selection algorithm. In: Grefenstette JJ (ed) Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, New Jersey, pp 14–21

Baker M, Buyya R, Laforenza D (2002) Grid and grid technologies for wide area distributed computing. Software: Practice and Experience 32(15):1437–1466

Berman F, Wolski R, Figueira S, Schopf J, Shao G (1996) Application-level scheduling on distributed heterogeneous networks. In: Supercomputing '96

Cantú-Paz E, Goldberg DE (1999) On the scalability of parallel genetic algorithms. Evolutionary Computation 7(4):429–449

Chappell DA, Jewell T (2002) Java Web Services. O'Reilly

Chipperfield AJ, Fleming PJ (1995) Parallel genetic algorithms. In: Zomaya AY (ed) Parallel And Distributed Computing Handbook, McGraw-Hill, chap 39, pp 1118–1144

Crocker J, Kumar UD (2000) Age-related maintenance versus reliability centred maintenance: A case study on aero-engines. Reliability Engineering and Systems Safety 67:113–118

Davis L (1985) Job shop scheduling with genetic algorithms. In: Grefenstette JJ (ed) Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, New Jersey, pp 136–140

Distributed Systems Architecture Group, Universidad Complutense de Madrid (2007) GridWay Metascheduler. URL `http://www.gridway.org/index.php`, viewed 22nd April 2007

Fernandez F, Tomassini M, Vanneschi L (2003) An empirical study of multipopulation genetic programming. Genetic Programming and Evolvable Machines 4:21–51

Fleming PJ, Purshouse RC, Chipperfield AJ, Griffin IA, Thompson HA (2002) Control systems desing with multiple objectives: An evolutionary computing approach. In: Workshops of the 15th IFAC World Congress

Fogarty TC, Huang R (1991) Implementing the genetic algorithm on transputer based parallel processing systems. In: Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature 1, Springer-Verlag, Berlin, Lecture Notes in Computer Science, vol 496, pp 145–149

Fogel DB, Ghoziel A (1997) A note on representations and variation operators. IEEE Transactions on Evolutionary Computation 1(2):159–161

Foster I, Kesselman C (1999) The Globus Toolkit. In: Foster I, Kesselman C (eds) The GRID: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, chap 11, pp 259–278

Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications 15(3):200–222

Foster I, Kesselman C, Nick JM, Tuecke S (2002) Grid services for distributed system integration. IEEE Computer 35(6):37 – 46

Fung CC, Li JB, Wong KW, Wang KP (2004) A java-based parallel platform for the implementation of evolutionary computation for engineering applications. International Journal of Systems Science 35(13-14):741–750

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Grosso PB (1985) Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model. PhD thesis, University of Michigan

Hamby DM (1994) A review of techniques for parameter sensitivity analysis of environmental models. Environmental Monitoring and Assessment 32:135–154

Hancock PJB (1994) An empirical comparison of selection methods in evolutionary algorithms. In: Fogarty TC (ed) Evolutionary Computing - AISB Workshop, Springer-Verlag, Berlin, Lecture Notes in Computer Science, vol 865, pp 80–94

Herrera J, Huedo E, Montero RS, Llorente IM (2005) A grid-oriented genetic algorithm. In: Sloot PMA, Hoekstra AG, Priol T, Reinefeld A, Bubak M (eds) Advances in Grid Computing: EGC 2005, Springer-Verlag, Lecture Notes in Computer Science, vol 3470, pp 315–322

Jensen MT (2003) Generating robust and flexible job shop schedules using genetic algorithms. IEEE Transactions on Evolutionary Computation 7(3):275–288

Kleinrock L (1975) Queueing Systems Volume 1: Theory. John Wiley & Sons, New York

Klienrock L (1969) UCLA press release. URL `http://www.lk.cs.ucla.edu/LK/Bib/REPORT/press.html`

Langdon WB (1995) Scheduling planned maintenance of the national grid. In: Fogarty TC (ed) Evolutionary Computing - AISB Workshop, Springer-Verlag, Berlin, Lecture Notes in Computer Science, vol 993, pp 132–153

Lewis R, Paechter B (2005) Application of the grouping genetic algorithm to university course timetabling. In: Raidl GR, Gottlieb J (eds) Proceedings of the Fifth European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP), Springer-Verlag, Berlin, Lecture Notes in Computer Science, vol 3448, pp 144–153

Lim D, Ong Y, Jin Y, Sendhoff B, Lee B (2007) Efficient hierarchical parallel genetic algorithms using grid computing. Future Generation Computer Systems 23:658–670

Mesghouni K, Hammadi S, Borne P (2004) Evolutionary algorithms for job-shop scheduling. International Journal of Applied Mathematics and Computer Science 14(1):91–103

Michalewicz Z, Fogel DB (2000) How to Solve It: Modern Heuristics. Springer, Berlin

Mühlenbein H, Schlierkamp-Voosen D (1993) Predictive models for the breeder genetic algorithm I: Continuous parameter optimization. Evolutionary Computation 1(1):25–49

Pratihar D, Deb K, Ghosh A (1999) A genetic-fuzzy approach for mobile robot navigation amongst moving obstacles. International Journal of Approximate Reasoning 20(2):145 – 172

Rivera W (2001) Scalable parallel genetic algorithms. Artificial Intelligence Review 16(2):153–168

Rolls-Royce (2002) Mearos model description version 8.31. Rolls-Royce Internal Document

Shenfield A (2007) Grid-enabled optimisation using evolutionary algorithms. PhD thesis, University of Sheffield

Sims K (1991) Artificial evolution for computer graphics. Computer Graphics 25(4):319–328

Song W, Ong YS, Ng HK, Keane A, Cox S, Lee BS (2004) A service-oriented approach for aerodynamic shape optimization across institutional boundaries. In: Proceedings of the 8th ICARCV Control, Automation, Robotics and Vision Conference, pp 2274–2279

Starkweather T, Whitley D, Mathias K (1991) Optimization using distributed genetic algorithms. In: Schwefel HP, Männer R (eds) Parallel Problem Solving from Nature 1, Springer-Verlag, Berlin, Lecture Notes in Computer Science, vol 496, pp 176–185

Tan KC, Tay A, Cai J (2003) Design and implementation of a distributed evolutionary computing software. IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews 33(3):325–338

Tanese R (1987) Parallel genetic algorithms for a hypercube. In: Proceedings of the Second International Conference on Genetic Algorithms (ICGA2), pp 177–183

Tanimura Y, Hiroyasu T, Miki M, Aoi K (2002) The system for evolutionary computing on the computational grid. In: Proceedings of the 14th International Conference on Parallel and Distributed Computing Systems, ACTA press, pp 39–44

The GridPP Project (2007) GridPP - UK Computing for Particle Physics Website. URL http://www.gridpp.ac.uk/, viewed 23 March 2007

The White Rose University Consortium (2007) The White Rose Grid Website. URL http://www.wrgrid.org.uk/index.html, viewed 23 March 2007

W3C Working Group (2004) Web services architecture. URL http://www.w3c.org/TR/ws-arch, viewed 18 October 2006

Xue G, Song W, Cox SJ, Keane A (2004) Numerical optimisation as grid services for engineering design. Journal of Grid Computing 2:223–238