

A distributed global optimiser applied to the design of a constellation performing radio-occultation measurements

Dario Izzo, Mihály Csaba Markót, Isabelle Nann

ESA – Advanced Concepts Team

e-mail: dario.izzo@esa.int

**15th AAS/AIAA Space Flight
Mechanics Conference**

Copper Mountain, Colorado January 23-27, 2005

AAS Publications Office, P.O. Box 28130, San Diego, CA 92198

A DISTRIBUTED GLOBAL OPTIMISER APPLIED TO THE DESIGN OF A CONSTELLATION PERFORMING RADIO-OCCULTATION MEASUREMENTS

Dario Izzo^{*}, Mihály Csaba Markót[†], Isabelle Nann[‡]

Advanced Concepts Team

In an engineering problem global optimisation is concerned with finding a solution that cannot be improved. A large number of techniques and approaches have been proposed to tackle this fundamental issue, but when it comes to apply them to complex designs they all fail to return the solution in a short time. This problem, connected with the often large dimension of the search space and often referred to as “the curse of dimensionality”, is the major obstacle in locating the global optimum to many interesting problems. Most of the research in the past years has been focused on improving the global search algorithm by making use of novel techniques or of some knowledge of the search space. The computing effort needed, though, struggles to become reasonable as soon as the problem considered is described in detail. A solution to the huge computing power needed could be that of distributing the global optimiser computations over a very large net of computers letting the server to choose the most appropriate global algorithm to be used on the basis of the results continuously received from the other computers. In this article a preliminary architecture for such a net is presented and its future development is discussed. The distributed computing environment is then used to perform a search in the very complex space of constellation geometries, trying to look for a good geometry to perform radio-occultation measurements of the Martian atmosphere. The results show the potentiality of such a tool to reduce the computing time needed to select an optimal design point.

INTRODUCTION

The idea of using the idle time of our computers processors to perform useful calculations is a very powerful one. The distributed computing environment SETI@Home is having a success that is well described by its statistics updated to the 16th of January 2005: 2,191,000 years of CPU time have already been evaluated by a community of 5,320,000 people. This is helping the data processing of the huge quantity of signals that have to be scanned to search for an hypothetical extraterrestrial coherent transmission. Other initiatives have also been very successful in biophysics (folding@home) for the study of the unfolding of molecules and in encryption problems (distributed.net). Any problem that requires a huge quantity of calculations

^{*} Research Fellow, Advanced Concepts Team, ESA\ESTEC, Noordwijk, The Netherlands.

[†] Research Fellow, Advanced Concepts Team, ESA\ESTEC, Noordwijk, The Netherlands.

[‡] Young Graduate Trainee, Advanced Concepts Team, ESA\ESTEC, Noordwijk, The Netherlands.

to be solved and whose solving procedure is not heavily dependent from each piece of computation may be well suited to be distributed on a net. Consider for example the deterministic evolution of a complex system over an extremely long time scale. Any algorithm designed to solve such a computationally heavy problem would advance the solution in such a way that, at each time-step, the result from the previous iteration is needed. There are, though, complex and lengthy problems in which the results of the other calculations are not needed at all or are just useful but not necessary. The problem that is being solved by the SETI@Home distributed computing environment certainly belongs to the first type requiring a relatively simple task allocation strategy for the central server. In this paper we show how the problem of finding the global optimal solution to a certain given problem may be usefully thought about as belonging to the second category, and might be distributed in a net reducing significantly the overall computational time. There is a large number of problems that use global optimisation algorithms relevant to the design of space missions. These range from the design of interplanetary trajectories with Multiple Gravity Assist and low-thrust arcs¹⁻³ to the design of efficient antennas⁴ to the multidisciplinary optimisation of the entire system design⁵ to many other important examples. The Advanced Concepts Team of the European Space Agency, under the ARIADNA scheme, has recently funded a study⁶⁻⁷ on advanced global optimisation techniques applied to mission design. The study, performed separately by research groups belonging to the University of Glasgow⁷ and by the University of Reading⁶, tested a large number of global optimisation techniques on different problems related to spacecraft trajectory design. The works are a first attempt to create a “multi-dimensional taxonomy based upon the characteristics and the complexity of the considered models”, and to determine some relation between the classification introduced and the global search approach that was found to be the most efficient. In this way the idea was born, within the Advanced Concepts Team, to build a global optimizer able to self-learn what global optimisation strategy is the most appropriate for a generic problem given in form of a black-box and to run it efficiently to find a solution to the problem. The possibility to distribute this task over a large net of computer is here discussed and some results are given in the case of the optimisation of a Martian constellation performing radio-occultation measurements of the atmosphere. The motivation to choose such a design case was to try to understand whether and how the existing and planned Martian orbiters could be used, during their “unused” life-time to perform some measurements of the highly uncertain Martian atmosphere. The efficiency of such a concept is strictly related to what a good geometry is, in terms of orbits, to perform such a task. We did not therefore aim at finding the optimal configuration, rather at searching for different good orbital geometries able to perform a great number of occultation measurement uniformly distributed on the planet surface.

THE GLOBAL OPTIMISATION PROBLEM

We here consider the rather generic problem:

$$\begin{aligned} \min \quad & \mathbf{f}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in D \end{aligned} \tag{1}$$

where $x \in R^n$ is a n-dimensional vector, $f : R^n \Rightarrow R$ is a continuous function and $D \equiv \{\mathbf{x} = [x_i] \mid l_i \leq x_i \leq u_i, g_j(x) \leq 0 \ i = 1..n, j = 1..m\}$ is a sub-set of R^n defined by the upper and lower bounds u_i, l_i and by the continuous constraint functions g_j . The continuity conditions introduced are not necessary, but they do guarantee straight forwardly that there

exists at least one solution to the problem above stated (classic Weierstrass Theorem). The construction of an algorithm able to solve a problem in the general form of eq.(1), has been faced by many in the past, and has given birth to a great number of techniques and strategies (see Neumaier⁸). We are here concerned with algorithms that perform only point evaluation of the objective function and that do not need any kind of global information on it. This way we may think to the objective function and to the non linear constraints as black boxes having only the current solution point as input. A great number of popular methods have been developed in the past to face this intriguing problem: Monte Carlo, Genetic Algorithm, Evolutionary Algorithm, Ant Colonies, Simulated Annealing, Cross-Entropy, Particle Swarm Optimisation, Differential Evolution, Direct, Multilevel Coordinate Search, Tabu Search, are only some of them. No method can be said to be better than another as anyone of the above quoted strategies, and of the many others not reported in here, will surely be able to outperform the others for a particular problem.

All the methods for global optimisation have one important feature in common: they need to evaluate the objective function a great number of times before converging to the optimal point. This, depending on the problem, slows down all the strategies as soon as the search space, and as a consequence the number of function evaluations, becomes significantly large. The issue, commonly referred to as the “curse of dimensionality”, makes global optimisation a perfect problem to be embedded into a distributed computing environment. Several researchers have presented some global optimisation algorithms able to be distributed as requiring minimal communication and planning between the various collaborative machines. Tsui and Liu¹² developed an Evolutionary Diffusion Optimisation algorithm suitable for distributed computing, whereas Colomi et al.¹³ discussed the Ant Colony search as a potentially distributed global optimiser. Lampinen¹⁴ introduces a master-slave architecture for distributed differential evolution and many other research groups are looking into grid computing technologies to allow this sort of computations. The approach here proposed, and the resulting preliminary architecture, is based on the idea to use many different stochastic and deterministic algorithms to evolve the same set of solutions and to instruct the server to allocate efficiently different tasks according to the results received along the optimisation. Each computer evolves the assigned population for an assigned number of generations or search in a given part of the space with a randomly selected algorithm. The chances of a particular algorithm to be selected and assigned as a task depend on the efficiency of the algorithm in the previous calls. The number of generations evolved and the parameters needed to tune a particular strategy have also to be considered in this selection process. At the end the aim is to obtain a global optimiser able to self-learn how to efficiently solve instances of a given problem class, a global optimizer that, exploiting the power of a distributed net, would be able to perform a thorough research of the best possible solution.

THE DISTRIBUTED NET ARCHITECTURE

The architecture of the present version of the distributed computing environment is demonstrated in Figure 1. The architecture has well-identified layers, which enhances the modular development of the whole system and promotes debugging and maintaining (e.g. the client computation side – the solvers - can be developed and tested independently from the layers to which it is connected). In the section we briefly discuss the main components and their functions.

The architecture follows the scheme of a generic server-client model⁹: it consists of a central computer (server) and a number of user computers (clients). The central computer is taking care of the whole computing process by performing the following tasks:

- pre-processing the whole computing task by disassembling it into subproblems,
- distributing sets of subproblems among the clients, and
- generating the final result of the computation by assembling the arrived sets of solutions.

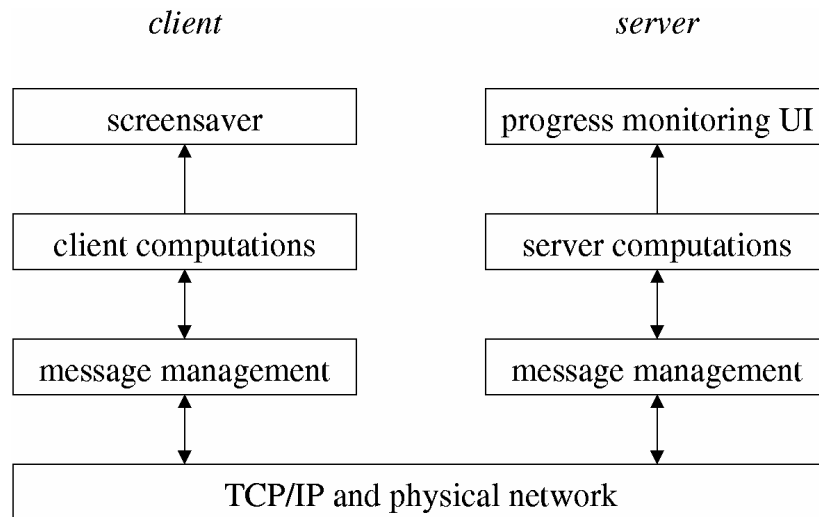


Figure 1: The distributed computing environment architecture

Thus, the main restriction of executing a given computing task in a distributed environment in an efficient way is that the task should be divided into well-defined smaller tasks, and these subproblems can be executed more or less independently of each other. As discussed above this is the case of the proposed global optimisation methodology.

The client computers ask for subproblems, solve them and send back the results to the server. As it is typical for distributed applications involving common user desktop machines, our approach is based on the utilization of the idle time of the client computers. This means that a client only asks for a subproblem when there is no user activity detected for a while (according to the use of the mouse and the keyboard). In our environment all the client activities are hidden behind a screensaver.

(Note, that since we detect only user activities on the peripherals, the computation process run concurrently with the possible background processes of the computer.) The idle-processing approach also involves that whenever the user is back to his work, the computation is interrupted and a sufficient answer (e.g. fractional but still usable parts of the expected solution) is sent back to the server.

As Figure 1 shows, the basic functionality of the individual layers is the same for both the client and the server point of view. The uppermost layers are visible for the client users and for the administrator of the computing project typically sitting behind the server. From the client side

this layer is the screensaver itself, from the server side this is a user interface monitoring the activities of the clients and displaying the actual state of the computation process.

These top-level layers communicate with the real computation layers performing the numerical tasks. The computation layers are responsible for disassembling, distributing and assembling the whole task (server), and executing the subtasks (clients). In our case study, presented later, the server-side computation layer generates *problem packages* consisting of requests to perform a given number of objective function evaluations, and stores the most promising solutions arrived - until a specific number of function evaluations is achieved. In the meantime, the client computation layer generates the sufficient number of random samples (by extracting the problem package), then it evaluates the objective function as many times as requested (or possible, in case the user deactivates the screensaver), and finally, it generates a *solution package* consisting of the most promising solution found and the number of function evaluations performed. This package forms the answer of the client.

It is easy to see that the concepts of problem, problem package, solution, and solution package can be generalized for many distributed computing tasks. Later in this section we will discuss the relations of these concepts within the present overall class structure. The specific roles of these concepts in the case of our experimental study will be explained in the last section.

The message management layers maintain connection with the computation layers, and send and receive the problem and solution packages between the client and the server. This service was implemented by *network sockets* using the Windows Sockets version 2 Application Programming Interface¹⁰. In particular, we applied connection-oriented sockets, i.e. we required that a proper connection is built up between the client and the server before performing any data transmission (in contrast to the so-called message-oriented sockets). Moreover, we used the sockets in blocking (synchronous) mode, that is, the socket functions (like 'connect' or 'receive') do not return until they complete their work. The reason of choosing the above socket types was that each client works in a very strict way: every time the client connects to the server, it informs the server about the previous results and its present availability, and expects further instructions (like "perform further tasks" or "the main computing task is solved, do not ask for more problems"). In any case, in our future studies we plan to investigate the possibility of applying message-oriented and asynchronous sockets as well.

Figure 2 shows the simplified C++ class diagram of the present initial version of the distributed computing environment. This structure is valid for both the server and the client application programs. To demonstrate the relations between the classes, we used a notation based on the Unified Modeling Language specification¹¹. A solid line connecting two classes indicates a simple relationship (e.g. through some service) between the classes. A hollow arrow pointing from class A to class B represents the inheritance relation, i.e. that "A is a B". A class name with italic typesetting represents an abstract class, that is, a class which itself is not used to instantiate objects. If a class A is connected to a class B with a solid diamond located next to the box of A, then B is said to be a component of A, i.e. "A has a B".

The currently used structure has an abstract base class called *Package*, which derives three classes **ProbPackage**, **SolPackage**, and **InfoPackage**, used to instantiate (create) concrete packages. **ProbPackage** is applied to store a set of tasks (problems) to be performed by a client in one step. On the contrary, **SolPackage** objects create and store the representation of the results ("solutions") of the currently processed problem package. **InfoPackage** is designed to send particular information between the client and the server, regarding for example the client

configuration or the progression of the overall process. At the moment this class is used only to transfer the CPU frequency of the client computer to the server.

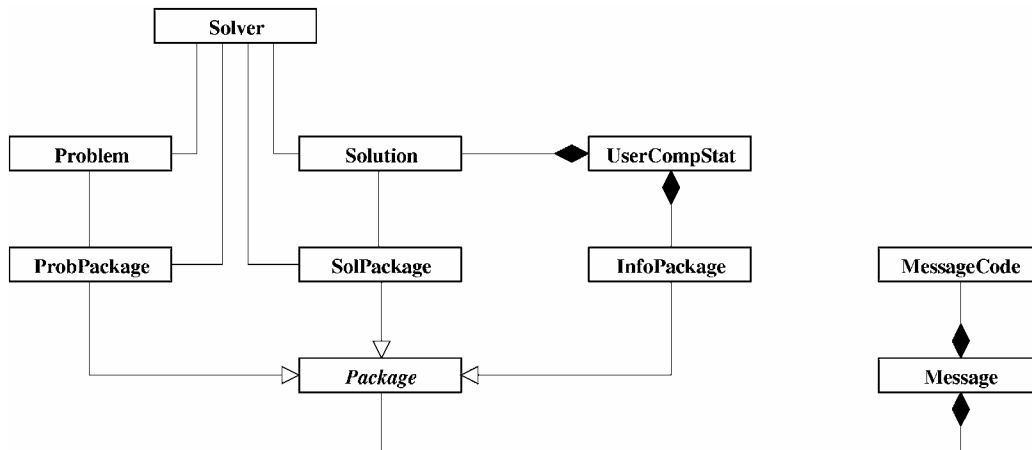


Figure 2: The C++ class structure

The classes **Problem** and **Solution** are related to the classes **ProbPackage** and **SolPackage** by the services of extracting/inserting one individual problem/solution from/to the corresponding package. An object of the class **Solver** acts on the above four classes.

The information transmission between the client and the server is based on the class **Message**. Each message consists of a particular package and a **MessageCode** containing instructions attached to the package (like “I am ready for further computations, send more problems to solve”). The server uses the **UserCompStat** class objects to store computation information related to the client machines, and also, to maintain summary information aggregating the individual client information. Currently **UserCompStat** is composed from an **InfoPackage**, a **Solution** (which is the current “best” solution found by the client), and several additional data fields and functions.

If one wants to use different solvers on the same problem than she has to make **Solver** to be an abstract class and to program the particular solvers as classes derived from **Solver**. It is worth to mention again that the present class structure is designed mainly for the purpose of our case study. During the future development of a flexible, multiple purpose distributed environment (applicable for solving both optimisation and other kind of numerically intensive problems) this structure will be further refined and extended.

THE RADIO-OCCULTATION PROBLEM

Evidence on the possibility to perform useful measurements of some planet atmospheric properties by means of inter-satellite links was first given in 1995 when a LEO-GNSS occultation measurements campaign was performed and used to improve the existing numerical models of the Earth atmosphere. The European Space Agency founded, in the last decade, a number of pre-phase A (e.g. the Atmosphere and Climate Explorer Plus ACE+) studies and

carried out the assessment of some innovative concepts (e.g. the Advanced Concepts Team reconfigurable Mars constellation) involving radio-occultation measurements. The design of a constellation that has to perform radio-occultation measurements of a planet atmosphere is an intriguing challenge as many different aspects have to be taken into account: the occultation length, their distribution upon the planet surface, the occultation direction and their number per day. Some early concepts (by ALCATEL and ALENIA spazio) propose the use of Walker constellations, or of the so called counter-rotating constellation concepts, but these geometries are only suboptimal and one should be able to do better. One of the greatest difficulties in obtaining an optimal configuration is the extremely complicated relation between the constellation geometry (i.e. the number of satellites and the set of their osculating parameters) and the occultation position, length and number. If we want to use some kind of optimal approach on this problem we must be able to assess a great number of different constellation geometries (the search space is in fact quite large) and we want to do this as quickly as possible. In order to avoid direct integration one may use the simple keplerian dynamic hypothesis noting that the disturbances do not affect much the “quality” of a given geometry. Even when all the possible simplifications are done the computation of the constellation performances can still be quite slow. The aim of the global optimisation performed was that of maximising the number of occultation per Martian day and their spatial distribution uniformity. Attention was paid not to the location of the global optimal but on finding good constellation geometries. In order to define some index describing the spatial distribution of the occultation we divided the Martian surface into N sectors comprised between two longitudes by taking care that:

$$\sin \lambda_{i+1} - \sin \lambda_{i-1} = 2 \sin \lambda_i$$

(so that the area of each sector is equal). For a given geometry the number of occultation n_i occurring in each sector was counted and the variance σ^2 was introduced:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (n_i - \bar{n})^2$$

Each geometrical configuration was therefore judged on the basis of the total number of occultation and of the value of the variance. In particular it was found that the use of an objective function with the form:

$$J(n, \sigma) = \frac{n}{\sigma^p}$$

where p is a positive integer, returns good results in terms of multi-objective optimisation. The complexity of the optimisation is directly related to the number of minima and to the size s^* of the basin of attraction of the global minimum⁶. In the radio occultation point the problem resulted to be of the most complex case. The case selected to test the global optimisation environment was that of a Martian constellation, so as to gain some insight on the possibility of using current and future orbiter missions to Mars to improve our knowledge on the atmosphere properties of the planet. An occultation was considered such if the actual occultation lasted more than 50 s. and happened at an altitude smaller than 300 km.

PRELIMINARY RESULTS

For a preliminary test of the proposed distributed network architecture only a basic random search algorithm was implemented using the classes above defined, and the problem packages sent to the different computers consisted of the request to evaluate the objective function for a given number of randomly chosen individuals. A case of a four satellite constellation was considered. The best individual found during a single package evaluation was returned to the server as a solution package who stored it in the population of best fit individuals. In this way it was possible to retrieve the whole final population of 25 fit individuals and to refine it via local gradient descend techniques. This offers the advantage of having different “design points” to be able to choose from. The computation was performed during a normal working day at the European Space and Technology Center (ESTEC) when the users where anyway using their computers normally and, as this was just a test phase, the program (hidden behind a simple screensaver) was installed on a limited number of Windows-XP machines, namely on 9. The optimisation was left running also overnight when all the computers were idle.

At the end of the test the server identified the name of the computer that actually found the best solution (giving credits to the user is one efficient way of convincing him to actually use the installed screen-saver) and produced other information summarized in table 1.

Test results	
Elapsed time	13 h 28 m 41 s
Virtual CPU frequency	12.8 GHz
Theoretical peak performance [§]	12.8 Gflops
Average objective function evaluation speed	0.019 s.
Number of problems sent	2555000
Number of problems received	2500857

Table 1: Test results

The virtual CPU frequency was evaluated as the maximum of the sum of the individual CPU frequency during the test. The difference between the number of problems sent and the number of problems received arises from users that stopped the screensaver by requesting back their computer with some mouse or keyboard activity, forcing the client to send to the server only a partial problem solution. It is also worth to note how the speed of the overall computation was found to be linear with the virtual CPU frequency.

An example of an individual belonging to the final population is given in figure 1 where the geometry is visualized together with the spatial distribution of the actual occultation points.

[§] According to double-precision floating-point computations.

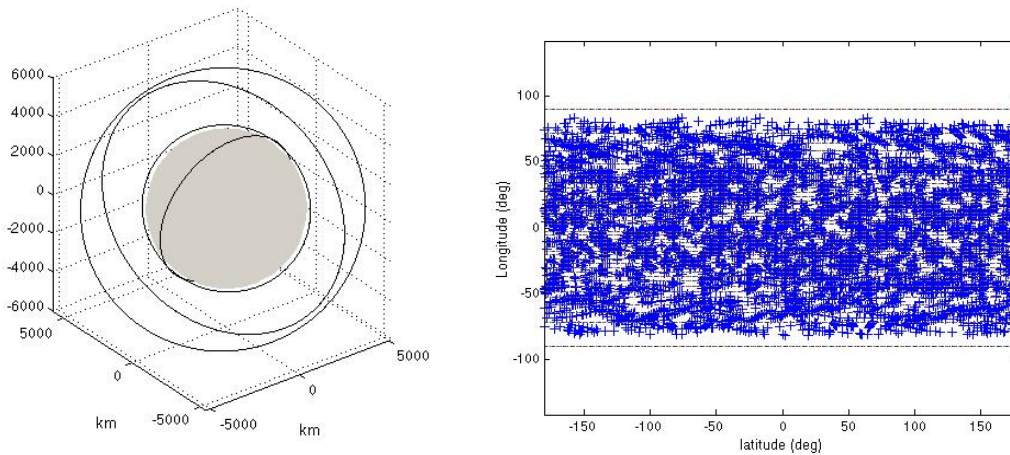


Figure 1: Example of a fit individual for the case of four Martian satellites. The number of occultations in two Martian days is in this case 1535 with a standard deviation of 7.78.

CONCLUSIONS

A distributed computing environment was created and operated in a small fraction of the internal European Space Agency network. The architecture was set up in order to be able to solve a global optimisation task via distributing different population evolving tasks based upon different numerical schemes. The distributed computing environment was tested to find good design points for a Martian constellation having to perform radio-occultation measurements of the atmosphere physical properties. In this preliminary test only Monte Carlo tasks were assigned proving the incredible increased computing power that such architecture may provide.

REFERENCES

1. R.Biesbroek, W.Ockels, G.Janin: “**Optimisation of Weak Stability Boundary Transfers from GTO to the Moon using Genetic Algorithms**”, Paper IAF-99-A.6.10, Amsterdam-Netherlands, 4-8 October 1999.
2. P.J.Gage, Braun R.D., Kroo I.M.: “**Interplanetary Trajectory Optimisation using a Genetic Algorithm**”. The Journal of the Astronautical Sciences, Vol. 43, No. 1, January-March 1995, pp. 59-75
3. B.Dachwald: “**Optimization of Interplanetary Solar Sailcraft Trajectories Using Evolutionary Neurocontrol**”. Journal of Guidance, Control, and Dynamics, Vol. 27, No. 1, 2004, pp. 66-72
4. F.Lattanzi: “**Joint system-antenna optimisation in the forward link of multi-beam unicast satellite systems utilising ACM**”, Tesi di Laurea Università di Roma di Tor Vergata, February 2005. Also available as final report of a stage period at ESTEC, Noordwijk.

5. G.B.Amata, G.Fasano, L.Arcaro, F.Della Croce, M.F.Norese, S.Palamara, R.Tadei, F.Fragnelli: “**Multidisciplinary Optimisation in Mission Analysis and Design Process**”, ESA GSP final report 03/N16, also available on-line at www.esa.int/gsp/ACT/studies_publications.htm
6. D.R.Myatt, V.M.Becerra, S.J.Nasuto, J.M.Bishop: “**Advanced Global Optimisation for Mission Analysis and Design**”, ESA ARIADNA final report 03/4101, also available on-line at www.esa.int/gsp/ACT/studies_publications.htm
7. P.Di Lizia, G.Radice: “**Advanced Global Optimisation for Mission Analysis and Design**”, ESA ARIADNA final report 03/4101, also available on-line at www.esa.int/gsp/ACT/studies_publications.htm
8. A. Neumaier: “**Complete Search in Continuous Global Optimization and Constraint Satisfaction**”, Acta Numerica., pp. 271-369, Cambridge University Press, 2004.
9. A.S. Tanenbaum: “**Computer Networks**”, Prentice Hall, 2003.
10. “**Windows Sockets 2 API documentation**”, [online document] <ftp://ftp.microsoft.com/bussys/winsock/winsock2/WSAPI22.DOC>
11. G. Booch, I. Jacobson, and J. Rumbaugh: “**Unified Modeling Language User Guide**”, Addison-Wesley, 1999.
12. K.C.Tsui, J.Liu: “**Multiagent Diffusion and Distributed Optimization**”, AAMAS conference, July 14-18, 2003, Melbourne, Australia.
13. A.Colorni, M.Dorigo, V.Maniezzo: “**Distributed Optimization by Ant Colonies**”, proceedings of the European Conference on Artificial Life, Paris, France, Elsevier pub. 134-142
14. J.Lampinen: “**Differential Evolution – New Naturally Parallel Approach for Engineering Design Optimization**”, in Barry H.V. Tooping (ed.)(1999) Developments in Computational Mechanics with High Performance Computing. Civil-Comp Press, Edinburgh (Scotland), pp.217-228