ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

# OPTIMIZATION ENVIRONMENTS AND THE NEOS SERVER

**William Gropp and Jorge J. Moré**

Mathematics and Computer Science Division

Preprint MCS-P654-0397

March 1997

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by a grant of Northwestern University to the Optimization Technology Center, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

# OPTIMIZATION ENVIRONMENTS AND THE NEOS SERVER

**William Gropp and Jorge J. Moré***

## 1 Introduction

In an ideal computational environment the user would formulate the optimization problem and obtain results without worrying about computational resources. Unfortunately this ideal environment is not possible because if sufficient care is not given to the formulation, a reasonable problem may become untractable. Even with an appropriate formulation, obtaining the solution of difficult optimization problems requires sophisticated optimization software and access to large-scale computational resources. Modeling three-dimensional physical processes by systems of differential equations gives rise to optimization problems that require access to substantial computational resources. Discrete and global optimization problems are also in this category.

We are interested in the development of problem-solving environments that simplify the formulation of optimization problems, and the access to computational resources. Once the problem has been formulated, the first step in solving an optimization problem in a typical computational environment is to identify and obtain the appropriate piece of optimization software. The software may be available from a mathematical software library, or may need to be bought and installed. In some cases the software is public domain, and available from a site on the Internet. Once the software has been installed and tested in the local environment, the user must read the documentation and write code to define the optimization problem in the manner required by the software. Typically, Fortran or C code must be written to define the problem, compute function values and derivatives, and specify sparsity patterns. Finally, the user must debug, compile, link, and execute the code.

The Network-Enabled Optimization System (NEOS) is an Internet-based service for optimization providing information, software, and problem-solving services for optimization. The main components of NEOS are the NEOS Guide and the NEOS Server. Additional information on the various services provided by NEOS can be obtained from the home page of the Optimization Technology Center

`http://www.mcs.anl.gov/home/otc/`

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844.
This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by a grant of Northwestern University to the Optimization Technology Center, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

The NEOS (Network-Enabled Optimization System) Server [11] is a novel environment for the solution of optimization problems that allows users to solve optimization problems over the Internet while requiring only that the user provide a specification of the problem. There is no need to download an optimization solver, write code to link the optimization solver with the optimization problem, or compute derivatives. Moreover, the NEOS Server provides an interface that is problem oriented and independent of the computing resources offered by NEOS. As long as there is an efficient way to describe the problem, the NEOS Server can provide access to a wide variety of computational services, from small clusters of workstations to any number of participating supercomputer centers.

The current version of the NEOS Server is described in Section 2. We emphasize nonlinear optimization problems, but NEOS does handle linear and nonlinearly constrained optimization problems, and solvers for optimization problems subject to integer variables are being added.

Performance issues are examined in Section 3. In particular, we provide evidence that the NEOS Server is able to solve large nonlinear optimization problems in time comparable to software with hand-coded gradients. We do not discuss the design and implementation of the Server because these issues are covered by Czyzyk, Mesnier, and Moré [11].

In Section 4 we begin to explore possible extensions to the NEOS Server by discussing the addition of solvers for global optimization problems. Section 5 discusses how a remote procedure call (RPC) interface to NEOS addresses some of the limitations of NEOS in the areas of security and usability. The detailed implementation of such an interface raises a number of questions, such as exactly how the RPC is implemented, what security or authentication approaches are used, and what techniques are used to improve the efficiency of the communication. These questions are not discussed here. Instead, we outline some of the issues in network computing that arise from the emerging style of computing used by NEOS.

## 2 The NEOS Server

The NEOS Server provides Internet access to a library of optimization solvers with user interfaces that shield the user from the optimization software. The user needs only to describe the optimization problem; all additional information required by the optimization solver is determined automatically.

The NEOS approach offers considerable advantages over a conventional environment for solving optimization problems. Consider, for example, how NEOS solves an optimization problem of the form

$$\min \{ f(x) : x \in \mathbb{R}^n \} ,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is partially separable, that is, $f$ can be written as

$$f(x) = \sum_{i=1}^{n_f} f_i(x),$$

where each element function $f_i$ only depends on a few components of $x$, and $n_f$ is the number of element functions. Algorithms and software that take advantage of partial separability have been developed for various problems (for example, [23, 24, 25, 26, 9]), but this software requires that the user provide the gradient of $f$ and the partial separability structure (a list of the dependent variables for each element function $f_i$).

The NEOS solvers for partially separable problems require that the user specify the number of variables `n`, a subroutine `initpt(n,x)` that defines the starting point, and a subroutine `fcn(n,x,nf,fvec)` that evaluates the element functions. Since there is no need to provide the gradient or the partial separability structure, the user can concentrate on the specification of the problem. Changes to the `fcn` subroutine can be made and tested immediately; the advantages in terms of ease of use are considerable.

The NEOS solvers for the bound constrained problem

$$\min \{ f(x) : x_l \le x \le x_u \}$$

and the nonlinearly constrained optimization problem

$$\min \{ f(x) : x_l \le x \le x_u, \;\; c_l \le c(x) \le c_u \}$$

also make use of partial separability. The bound constrained problem is specified by a subroutine that specifies the bounds $x_l$ and $x_u$, while for the nonlinearly constrained problem we also need to specify a subroutine that specifies the constraint bounds $c_l$ and $c_u$, and the nonlinear function $c : \mathbb{R}^n \mapsto \mathbb{R}^m$. Specifying this information is done with additional subroutines. The bounds $x_l$ and $x_u$ are specified with the subroutine `xbound(n,xl,xu)`, the constraint bounds $c_l$ and $c_u$ are specified with the subroutine `cbound(m,cl,cu)`, and the nonlinear function $c : \mathbb{R}^n \mapsto \mathbb{R}^m$ is specified by `cfcn(m,x,c)`.

We have mentioned nonlinear optimization solvers, but NEOS contains solvers in other areas. At present we have solvers in the following areas:

Unconstrained optimization
Bound constrained optimization
Nonlinearly constrained optimization
Complementarity problems
Linear network optimization
Linear programming
Stochastic linear programming

The addition of solvers in other areas is not difficult; indeed, NEOS was designed so that solvers in a wide variety of optimization areas could be added easily.

We provide Internet users the choice of three interfaces for submitting problems: e-mail, the NEOS Submission tool, and the NEOS Server Web interface. The interfaces are designed so that problem submission is intuitive and requires the minimal amount of information. The interfaces differ only in the way that information is specified and passed to the NEOS Server.

The e-mail interface is relatively primitive, but useful because most users have easy access to e-mail. Information on the available services and on the format used to submit problems via e-mail can be obtained by sending the mail message help to

`neos@mcs.anl.gov`

Users interested in the Web interface should visit the URL

`http://www.mcs.anl.gov/home/otc/Server/`

This URL has links to all the solvers in the library, as well as pointers to other NEOS information, in particular, the NEOS Guide. In the remainder of this section we examine the NEOS Submission tool.

The NEOS Submission tool provides a high-speed link to the NEOS Server via sockets. Once this tool is installed, the user has access to all the services provided by the NEOS Server. Users may download the Submission tool from the URL

`http://www.mcs.anl.gov/otc/Server/submission_tool.html/`

Installation of the Submission tool is immediate provided that Perl [28] has been installed properly. If the installation fails, the usual remedy is to run the Perl `h2ph` script that changes C header files into Perl header files. Running the `h2ph` script is simple but must be done by the installer of Perl, which is usually the system administrator.

Submission of problems via the NEOS Submission tool is simple. The user must first choose the type of optimization problem. Once an area is selected, the user must choose a solver. This selection process is done via drop-down menus typical of well-designed user interfaces.

The optimization problem is specified via a submission form. For example, Figure 2.1 shows the NEOS Submission form for the vmlm solver of unconstrained optimization problems. Solvers in each area have a submission form that is appropriate for that area.

For the vmlm solver the user needs to specify the language used to submit the problem (Fortran or C), the number of variables $n$, the number of partially separable functions $n_f$, and the files for the initial point and function evaluation subroutines. Browse buttons are available to ease the specification of the various files. An advantage of this interface is that,

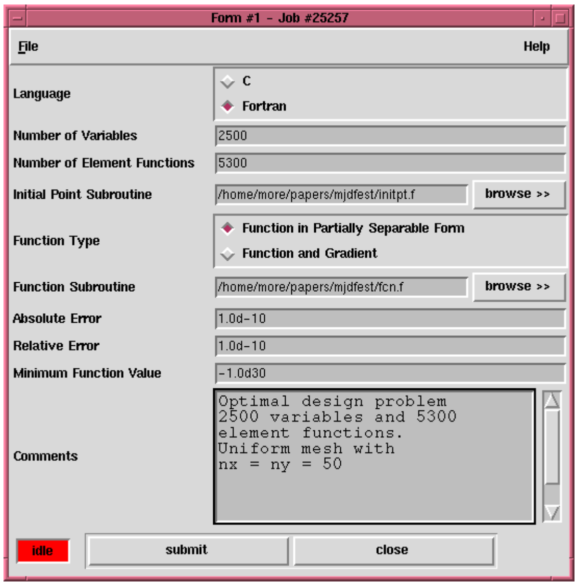


Figure 2.1: The NEOS submission form for vmlm

unlike the Web interface, the subroutines can be in files that reside in the user's local file space.

The general philosophy of the NEOS solvers is that problem submission should be intuitive and require only essential information. Parameters that affect the progress of the algorithm are not required but can be specified, for example, by a specification file. The vmlm solver allows the user a choice of tolerances, but for most problems the defaults provided are adequate. The form also has room for comments, which can be used to identify the problem submission.

Once the problem is specified, the problem is submitted via the submission button at the bottom of the form (see Figure 2.1). A variety of computers, even a massively parallel processor, could be used to solve the problem; the only restriction is that the computer must run UNIX with support for TCP/IP. At present these computers are workstations that reside at Argonne National Laboratory, Northwestern University, and the University of Wisconsin.

For a typical submission, the user receives information on the progress of the submission, and the solution. Figure 2.2 shows part of the output received when the problem in Figure 2.1 is submitted to NEOS. This output shows that NEOS contacts an available workstation and transfers all of the user's data to the workstation. The solver (in this case vmlm) checks the data and compiles the user's code. If any errors are found at this stage, the compiler error messages are returned to the user, and execution terminates.

If the user's code compiles correctly, automatic differentiation tools (ADIFOR [4, 3] for Fortran code) are used to generate the gradient. Once the gradient is obtained, the user's

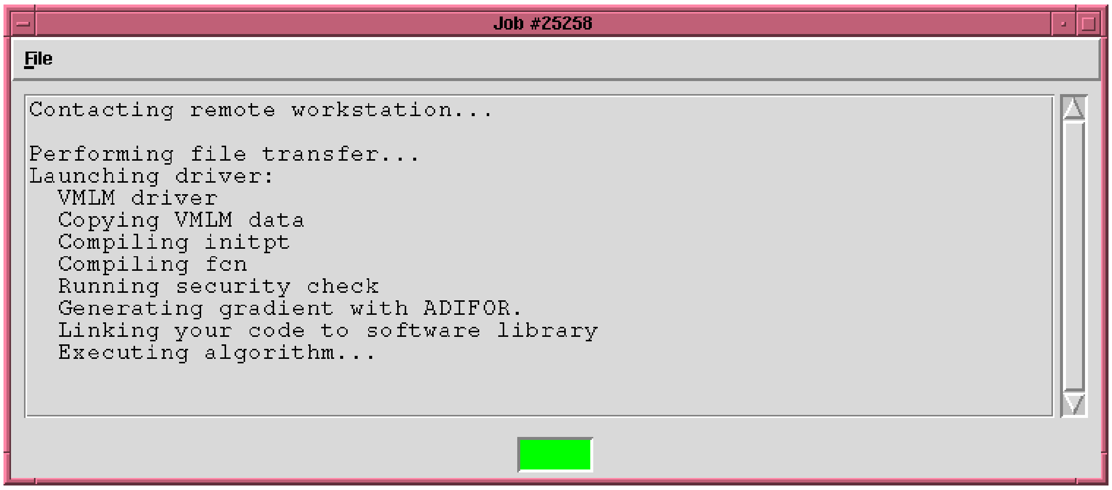


Figure 2.2: Output from the NEOS Submission tool

code is linked with the software library, and execution begins. Results are returned in the window generated by the NEOS Submission tool.

Interesting issues arise during the processing of the job submission that are pertinent to the development of optimization software and problem-solving environments. For example, high-quality software should check the input data, but in this case the data are the Fortran programs initpt and fcn. In general, it is not possible to check that this data is correct. At present we check only that the user function does not create any system exceptions during the evaluation of the function at the starting point. Although simple, this test catches many errors on the part of the user.

Submitting a problem to the NEOS Server does not guarantee success, but NEOS users are able to solve difficult optimization problems without worrying about many of the details that are typical in a computing environment. Even if the user has suitable optimization software, the user would need to read the documentation, write code to interface his problem with the optimization software, and then debug this code. The user would also have to write code for the gradient, and debug that code–a nontrivial undertaking in most cases.

## 3 Performance

The NEOS solvers for partially separable problems are able to solve large-scale nonlinear optimization problems while requiring only that the user provide code for the function evaluation. This ability was considered unrealistic until recently. The major obstacle was the computation of the gradient. For small-scale problems we can approximate the gradient by differences of function values, for example,

$$[\nabla f(x)]_i \approx \frac{f(x + h_i e_i) - f(x)}{h_i}, \qquad 1 \le i \le n,$$

where $h_i$ is the difference parameter and $e_i$ is the $i$th unit vector, but this approximation is prohibitive for large-scale problems because it requires $n$ function evaluations for each gradient. Approximating a gradient by differences is not only expensive but also increases the unreliability of the optimization code, since a poor choice for $h_i$ may cause premature termination of the optimization algorithm far away from the solution.

The NEOS solvers for nonlinear optimization problems use automatic differentiation tools to compute the gradients, Jacobians, and sparsity patterns required by the solvers. At present, we rely on ADIFOR [4, 3] to process Fortran code and on ADOL-C [15] to process C code.

We demonstrate the ability of NEOS to solve large-scale nonlinear optimization problems with an optimal design problem formulated by Goodman, Kohn, and Reyna [14]. This optimal design problem requires determining the placement of two elastic materials in the cross section of a rod with maximal torsional rigidity. The mathematical formulation is to minimize a functional of the form

$$f_\lambda(v) = \int_{\mathcal{D}} \left\{ \psi_\lambda \left( \|\nabla v(x)\| \right) + v(x) \right\} dx,$$

over a domain $\mathcal{D}$ in $\mathbb{R}^2$, where $\psi_\lambda : \mathbb{R} \mapsto \mathbb{R}$ is a piecewise quadratic. The formulation of the optimal design problem with finite elements leads naturally to a partially separable optimization problem in $n = n_x n_y$ variables, where $n_x$ and $n_y$ are the number of interior grid points in the coordinate directions, respectively. We use the formulation in the MINPACK-2 test problem collection [1]. Additional details on the problem formulation are not important to our discussion. We need to know only that in our numerical results we consider the problem of minimizing $f_\lambda$ for a fixed value of $\lambda$; in this case $\lambda = 0.008$.

From a computational viewpoint, the most interesting feature of the code to evaluate $f_\lambda$ is that the number of floating-point operations required to evaluate $f_\lambda$ grows linearly with $n$. Ideally, we would like to solve the problem in time proportional to $n$.

We solve the optimal design problem by developing code to evaluate $f_\lambda$. In our formulation the vector `x` contains the values of the piecewise linear finite element approximation, and the subroutine

```
dodc(nx,ny,x,nf,fvec,lambda)
```

evaluates the components of the partially separable function $f_\lambda$ as a function of the number of grid point and $\lambda$. The components of the partially separable function are stored in the array `fvec` of length `nf`. In this formulation `nf` is the number of elements in the finite element triangulation.

Ths subroutine `dodc` does not have the desired form for submission to NEOS, but it is quite easy to write a wrapper. For example, the results in this section were obtained with a subroutine of the form

```
fcn(n,x,nf,fvec)
```

that sets `nx` and `ny` to $n^{1/2}$ and $\lambda$ to 0.008. With this formulation we can quickly submit a series of problems to NEOS for various values of $n$.

Submission of the optimal design problem with the NEOS Submission tool is quite easy. Figure 2.1 shows the form that was used to submit the optimal design problem. In Figure 2.1 we were using $n = 2500$, but the form can be used for other values of $n$ by changing the number of variables and the number of elements functions.

Table 3.1 shows the timings (in seconds) and the number of function evaluations needed to solve an optimal design problem with the vmlm solver. We provide information for the case when the user only provides the function in partially separable form and for the case when the user provides the function and gradient. These results were obtained on a Sparc 10 with 96MB of memory.

Table 3.1: Performance of the NEOS solver vmlm

| | Function | | | Function and Gradient | | |
|---|---|---|---|---|---|---|
| $n$ | `niters` | `nfgev` | `time` | `niters` | `nfgev` | `time` |
| 2,500 | 230 | 237 | 139 | 232 | 239 | 22 |
| 10,000 | 427 | 436 | 1042 | 427 | 433 | 165 |
| 40,000 | 865 | 871 | 8399 | 877 | 885 | 1461 |

There are two important points to notice in the results in Table 3.1. The main point is that these results show that the time per function evaluation increases linearly with $n$. This is to be expected for this problem when the user provides both the function and the gradient, but it is remarkable that this also holds for the case when the user only provides the function. The techniques [2] used to achieve these results make essential use of the partial separability of the function.

Another important point about the results in Table 3.1 is that there is a factor of six penalty in the timings when only providing the function. If we had used a standard difference approximation to the gradient, there would have been a performance penalty of about $n$, which is prohibitive for these problems. We also note that for these results, vmlm used ADIFOR with the sparse option. This strategy is far from optimal; with the hybrid strategy of Bouaricha and Moré [7] the performance penalty is reduced to a factor of two.

Finally, we note that the number of function evaluations needed to solve the problem grows as a function of $n^{1/2}$. However, this is all that can be expected from a limited-memory variable metric method.

The main point that should be drawn from the results in Table 3.1 is that the NEOS Server combines an intuitive user interface, automatic differentiation tools, and optimization algorithms into a powerful problem-solving tool. We want to improve the NEOS Server by extending the range of problems that can be solved, but we also want to improve the interface. These issues will be examined in the next two sections.

## 4 Global Optimization Problems

We want to extend the NEOS Server capabilities to global optimization problems. In general, these problems are attacked by algorithms that require a large number of loosely coupled processors. We are interested, in particular, in problems that arise in connection with the determination of protein structures. As a specific instance of the type of problem under consideration, we consider distance geometry problems.

Distance geometry problems for the determination of protein structures are specified by a subset $\mathcal{S}$ of all atom pairs and by the distances between atoms $i$ and $j$ for $(i, j) \in \mathcal{S}$. In practice, lower and upper bounds on the distances are given instead of precise values. The distance geometry problem with lower and upper bounds is to find a set of positions $x_1, \ldots, x_m$ in $\mathbb{R}^3$ such that

$$l_{i,j} \le \|x_i - x_j\| \le u_{i,j}, \qquad (i, j) \in \mathcal{S}, \tag{4.1}$$

where $l_{i,j}$ and $u_{i,j}$ are lower and upper bounds on the distances, respectively. Recent reviews of the application of distance geometry problems to protein structure determination can be found in Havel [17, 18], Torda and van Gunsteren [27], Kuntz, Thomason, and Oshiro [19], Brünger and Nilges [8], and Blaney and Dixon [5].

A standard formulation of the distance geometry problem (4.1), suggested by Crippen and Havel [10], is to find the global minimum of the function

$$f(x) = \sum_{i,j \in \mathcal{S}} p_{i,j}(x_i - x_j),$$

where the pairwise function $p_{i,j} : \mathbb{R}^n \mapsto \mathbb{R}$ is defined by

$$p_{i,j}(x) = \min{}^2 \left\{ \frac{\|x\|^2 - l_{i,j}^2}{l_{i,j}^2}, 0 \right\} + \max{}^2 \left\{ \frac{\|x\|^2 - u_{i,j}^2}{u_{i,j}^2}, 0 \right\}.$$

Clearly, $x = \{x_1, \ldots, x_m\}$ solves the distance geometry problem if and only if $x$ is a global minimizer of $f$ and $f(x) = 0$.

Distance geometry problems can be specified by a small amount of data. A reasonable specification is to provide the number of atoms $m$, the size of the set $\mathcal{S}$, and the vectors

$$(i, j, l_{i,j}, u_{i,j}), \qquad (i, j) \in \mathcal{S}. \tag{4.2}$$

In a practical setting this data would be accompanied by additional information (for example, the type of atoms), but for our purposes we would need only (4.2).

Finding a global minimizer of $f$ for the distance geometry problem is NP-hard even in the special case where $l_{i,j} = u_{i,j}$. Finding approximate minimizers is also NP-hard; that is, the decision problem of finding $x \in \mathbb{R}^n$ such that $f(x) \le \epsilon$ for given $\epsilon > 0$ is NP-hard. At a more practical level, we note that the function $f$ has a large number of distinct

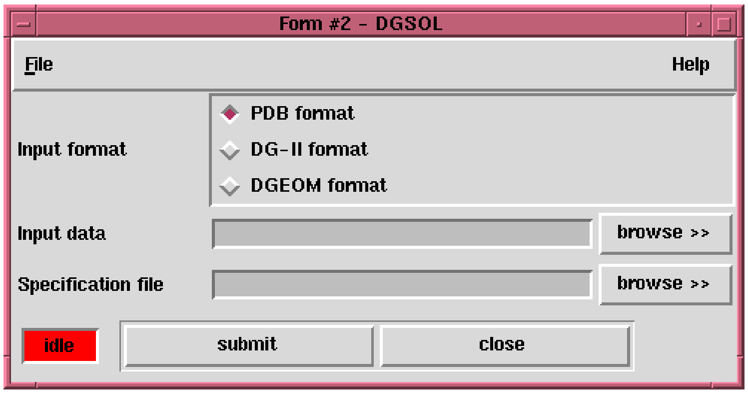


Figure 4.3: The NEOS submission form for dgsol

minima, which seem to grow exponentially with the number of atoms $m$. Moreover, in our experience, even problems with a small number of atoms can be quite difficult. See Moré and Wu [21, 22] for additional information.

Several general approaches to the solution of global optimization problems could be used to solve distance geometry problems: multi start techniques, simulated annealing, and genetic algorithms. There are also specific techniques for distance geometry problems, in particular, the embed algorithm (see Crippen and Havel [10], and Havel [17, 18]), and the dgsol algorithm of Moré and Wu [22].

We have designed the NEOS Server so that additional solvers can be installed easily, with little intervention by the Server administrators. Given a solver, the user is given an interface that is intuitive and simple. For distance geometry problems we could use, for example, the submission form in Figure 4.3 that requires the distance geometry data (4.2).

Registering a new solver with the NEOS Server automatically updates the library. Submission forms, in particular, are created from configuration files specified by the software administrator. Each line of the configuration file specifies an entry in the submission form. The form in Figure 4.3, for example, requires only a three-line configuration file. For additional details on the registration process, see Czyzyk, Mesnier, and Moré [11].

The main difficulty that must be faced in extending the NEOS Server to global optimization problems is the scheduling of jobs. With the current design, jobs are allocated sequentially to the first available machine, but we could use Condor [20], a distributed resource management system developed at the University of Wisconsin, to schedule jobs on heterogeneous clusters of workstations. NEOS is already using Condor to solve nonlinear complementarity problems (see Ferris, Mesnier and Moré [12]), so the transition to global optimization problems is feasible. Other research projects that are addressing the use of large-scale computing resources include Globus [13] and Legion [16].

The approach to solving global optimization problems with NEOS offers considerable advantages. Solving global optimization problems in a large-scale computing center would force the user to learn scheduling policies, job submission methods, and installation-specific software libraries. To some extent, such requirements are unavoidable; each center represents a significant investment and often must serve a particular user community. Even for users who must use these centers, the significant effort involved in learning the idiosyncrasies of each facility often limit the ability to switch between centers. In contrast, the NEOS Submission tool provides an interface that is problem oriented and independent of the computing resources that are offered by the NEOS Server.

## 5 Distributed Computing

The NEOS Server is first and foremost a network resource for optimization services. The current system provides convenient access to the facilities of NEOS, and, as described in Section 4, additional capabilities can be added easily. The current version has some limitations, however, in the area of of security and usability for computing in a distributed environment. Because the potential solutions to these problems are related, we discuss them together.

In any networked or distributed computing system, two important questions must be answered: How does the system protect itself from malicious or careless users (and these users from each other), and how does it present itself to the users? The first issue is one of security; the second is one of ease of use. NEOS currently has only rudimentary security features, and the interfaces do not allow access from a user program. A more traditional but more flexible and powerful method for accessing numerical capabilities is the procedural (subroutine) interface. The network analogue to this interface, the remote procedure call (RPC) [6], is useful for some parts of the security question as well. To better understand the security issues, we first outline the problem.

To make the discussion concrete, we consider the case of determining a minimizer of a function $f : \mathbb{R}^n \mapsto \mathbb{R}$. The most precise description of this problem is often the code that implements $f$. This code most likely contains loops and array accesses, and possibly function calls. If the NEOS Server simply accepts arbitrary code to describe the optimization problem, the systems running NEOS are subject to some security risks.

The most common security risk occurs if the user is not trusted. Lack of trust does not mean that the user is malicious, just that the work they send cannot be trusted not to cause problems. In this case we identify two issues: rogue programs and denial-of-service attacks.

Rogue programs are a security issue because the description of the optimization problem may crash the Server or even compromise the system. For problems (for example, linear programming or distance geometry) that are described by relatively simple data, it is easy to check that the data is valid. For programs, such checking is much harder. Our current system applies a number of techniques to reduce the chance for harm to the system. For example, the NEOS Server runs as a separate user with no special privileges. However, since the user can submit Fortran or C programs, it is possible that a programming error or a deliberate subterfuge could harm NEOS or, through some unplugged security problem, compromise the entire system. Solutions to this problem range from provably safe problem descriptions (either a specialized modeling language or strongly typed languages such as Java) to provably safe interfaces, with user code and NEOS code running on different systems.

Another security issue arises, even with well-meaning users, when programs use large amounts of computing—more than their fair share. When this is done maliciously, it is called a denial-of-service attack. The amount of time used by a single request is relatively easily controlled, and most scheduling systems provide this service. Harder to contain are floods of requests, each generating a separate request. The NEOS Server may need to maintain records of use as well as interacting with the scheduling system to ensure that the Server does not make excessive demands on the system.

A security risk also occurs if the user is trusted but the connection is not, because there may be eavesdroppers. This requires merely that some secure transaction service be used. Such services are becoming commonplace on the Web (for example, for credit card transactions), though usually for small amounts of data.

Providing both a more secure execution environment for NEOS and a more flexible interface makes use of the same technique: the remote procedure call. Conventionally, programmers of scientific applications call library routines from their Fortran or C program to perform some service, such as minimizing a function. For example, the user may call a procedure of the form

```
call uco( fcn, x, ... )
```

to find a minimizer of $f$. A remote procedure call looks just like a regular procedure call to the user; the implementation, however, makes contact with a remote server (NEOS in our case) and passes to the server the arguments of the call. The implementation may look like

```
call neos_uco( 'fcn.f', x, ... )
```

where `'fcn.f'` is the name of the file containing the code that defines the function $f$.

A remote procedure call interface would be a significant improvement over the interface provided by the NEOS Submission tool because such an interface would make it possible to incorporate NEOS capabilities into existing code. The remote procedure call interface is a relatively simple addition to the existing NEOS services that is secure and easy to use for those optimization problems that are completely described by simple data, such as linear programming problems. The data can be checked for validity by the server; encryption can be used if necessary to keep the data safe from eavesdroppers. If the data is Fortran or C code, however, checking for validity is far more difficult. If the user is not trusted, we can use a reverse communication approach or a provably safe form of code.

In the reverse communication approach, the user's function is executed on the user's machine. The RPC mechanism is used to request information from the user and to return information to the Server. For example, the user could be requested to evaluate the function $f$ at the current iterate $x$ and return $f(x)$ to the Server. This maintains the security against rogue code by executing the user's code on the user's machine and the optimization code on the NEOS-provided machine.

A drawback of this approach is that the cost of each of these remote procedure calls can be large. Just the delay from speed-of-light propagation can amount to milliseconds in a widely distributed network. In view of this limitation, this approach would be suitable only for problems in which the function evaluations requires considerable computing power so that the cost of the remote procedure call is insignificant.

The option to use a provably safe form of the code has attracted attention because of Java, a language related to C++ that contains a number of features and restrictions to allow programs written in Java to be executed without fearing that the system running them could be compromised. Many of the restrictions are irrelevant to most numerical code; for example, Java has no function pointers and restricts the kinds of routines (other than those provided by the user) that can be called.

In the most obvious approach, the user writes their code in Java instead of Fortran or C, and the Java code is uploaded into the NEOS Server by the remote procedure call. There is another option; the optimization code provided by NEOS could be downloaded directly into the user's running application (as long as that application is running in a Java environment). The interface would be the same; only the source of the cycles used to run the optimization code would be different. In this model, the procedure interface is a sort of automatic download-and-install operation that would be suitable only for small problems, and would not provide access to large-scale computing resources.

Java is not the only choice; a suitable subset of Fortran could be constructed by restricting some features of the language and performing strict compile and runtime checking. Such a network-safe version of Fortran could provide easy and reliable access to distributed services to existing programs.

## Acknowledgments

The ideas discussed in this paper have evolved during the development of the NEOS Server. Many have contributed to this effort, but Michael Mesnier deserves special credit. Other researchers that have contributed to these ideas are Joe Czyzyk, Michael Ferris, Jorge Nocedal, and Steve Wright.

## References


[1] B. M. Averick, R. G. Carter, J. J. Moré, and G.-L. Xue, *The MINPACK-2 test problem collection*, Preprint MCS-P153-0694, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[2] C. Bischof, A. Bouaricha, P. Khademi, and J. J. Moré, *Computing gradients in large-scale optimization using automatic differentiation*, Preprint MCS-P488-0195, Argonne National Laboratory, Argonne, Illinois, 1995. To appear in INFORMS Journal on Computing.

[3] C. Bischof, A. Carle, and P. Khademi, *Fortran 77 interface specification to the SparsLinC library*, Technical Report ANL/MCS-TM-196, Argonne National Laboratory, Argonne, Illinois, 1994.

[4] C. Bischof, A. Carle, P. Khademi, and A. Mauer, *The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs*, Preprint MCS-P381-1194, Argonne National Laboratory, Argonne, Illinois, 1994. Also available as CRPC-TR94491, Center for Research on Parallel Computation, Rice University.

[5] J. M. Blaney and J. S. Dixon, *Distance geometry in molecular modeling*, in Reviews in Computational Chemistry, K. B. Lipkowitz and D. B. Boyd, eds., vol. 5, VCH Publishers, 1994, pp. 299–335.

[6] J. Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc., 1992.

[7] A. Bouaricha and J. J. Moré, *Impact of partial separability on large-scale optimization*, Preprint MCS-P487-0195, Argonne National Laboratory, Argonne, Illinois, 1995. Accepted for publication in Computational Optimization and Applications.

[8] A. T. Brünger and M. Nilges, *Computational challenges for macromolecular structure determination by X-ray crystallography and solution NMR-spectroscopy*, Q. Rev. Biophys., 26 (1993), pp. 49–125.

[9] A. R. Conn, N. I. M. Gould, and P. L. Toint, *LANCELOT*, Springer Series in Computational Mathematics, Springer-Verlag, 1992.

[10] G. M. Crippen and T. F. Havel, *Distance Geometry and Molecular Conformation*, John Wiley & Sons, 1988.

[11] J. Czyzyk, M. P. Mesnier, and J. J. Moré, *The Network-Enabled Optimization System (NEOS) Server*, Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.

[12] M. C. Ferris, M. P. Mesnier, and J. J. Moré, *The NEOS Server for complementarity problems: PATH*, Technical Report 96-08, University of Wisconsin, Madison, Wisconsin, 1996. Also available as MCS-P616-1096, Mathematics and Computer Science Division, Argonne National Laboratory.

[13] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications, (1997). To appear.

[14] J. Goodman, R. Kohn, and L. Reyna, *Numerical study of a relaxed variational problem from optimal design*, Comput. Methods Appl. Mech. Engrg., 57 (1986), pp. 107–127.

[15] A. Griewank, D. Juedes, and J. Utke, *ADOL-C: A package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, 22 (1996), pp. 131–167.

[16] A. Grinshaw and W. Wolf, *Legion - A view from 50,000 feet*, in Proceedings of the 5th IEEE Symposium on High Performance Distributed Computing, Los Alamitos, California, 1996, IEEE Computer Society Press, pp. 89–99.

[17] T. F. Havel, *An evaluation of computational strategies for use in the determination of protein structure from distance geometry constraints obtained by nuclear magnetic resonance*, Prog. Biophys. Mol. Biol., 56 (1991), pp. 43–78.

[18] ——, *Distance geometry*, in Encyclopedia of Nuclear Magnetic Resonance, D. M. Grant and R. K. Harris, eds., John Wiley & Sons, 1995, pp. 1701–1710.

[19] I. D. Kuntz, J. F. Thomason, and C. M. Oshiro, *Distance geometry*, in Methods in Enzymology, N. J. Oppenheimer and T. L. James, eds., vol. 177, Academic Press, 1993, pp. 159–204.

[20] M. J. Litzkow, M. Livny, and M. W. Mutka, *Condor - A hunter of idle workstations*, in Proceedings of the 8th International Conference on Distributed Computing Systems, Washington, District of Columbia, 1988, IEEE Computer Society Press, pp. 108–111.

[21] J. J. Moré and Z. Wu, *$\varepsilon$-optimal solutions to distance geometry problems via global continuation*, in Global Minimization of Nonconvex Energy Functions: Molecular Conformation and Protein Folding, P. M. Pardalos, D. Shalloway, and G. Xue, eds., American Mathematical Society, 1995, pp. 151–168.

[22] ——, *Distance geometry optimization for protein structures*, Preprint MCS-P628-1296, Argonne National Laboratory, Argonne, Illinois, 1996.

[23] P. L. Toint, *Numerical solution of large sets of algebraic nonlinear equations*, Math. Comp., 46 (1986), pp. 175–189.

[24] ———, *On large scale nonlinear least squares calculations*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 416–435.

[25] P. L. Toint and D. Tuyttens, *On large-scale nonlinear network optimization*, Math. Programming, 48 (1990), pp. 125–159.

[26] ———, *LSNNO: A Fortran subroutine for solving large-scale nonlinear network optimization problems*, ACM Trans. Math. Software, 18 (1992), pp. 308–328.

[27] A. E. Torda and W. F. van Gunsteren, *Molecular modeling using nuclear magnetic resonance data*, in Reviews in Computational Chemistry, K. B. Lipkowitz and D. B. Boyd, eds., vol. 3, VCH Publishers, 1992, pp. 143–172.

[28] L. Wall, T. Christiansen, and R. L. Schwartz, *Programming Perl*, O'Reilly & Associates, Inc., second ed., 1996.