Educational article

Struct Multidisc Optim 22, 179–187 © Springer-Verlag 2001

A web-based topology optimization program

D. Tcherniak and O. Sigmund

Abstract The paper presents a web-based interface for a topology optimization program. The program is accessible over the World Wide Web at the address `http://www.topopt.dtu.dk`. The paper discusses implementation issues and educational aspects as well as statistics and experience with the program.

Key words topology optimization, web-based programming, engineering education

1 Introduction

In its simplest form, the topology optimization method solves the problem of distributing a given amount of material in a design domain subject to load and support conditions, such that the stiffness of the structure is maximized. Since its introduction (Bendsøe and Kikuchi 1988), the method has gained widespread popularity in academia and is now being applied to the design of materials, mechanisms, MicroElectroMechanical Systems (MEMS) and many other complex structural design problems.

The application of the topology optimization method in various fields of engineering can significantly improve design cost and quality which is important in global competition. Several papers have reported industrial applications of topology optimization such as design of automotive and airplane structures, however, the method, despite its attractiveness, is still not well-known in wide circles of practising design engineers, university students and engineering companies. The same can be said for design optimization in general (Hinton and Toropov 1999).

From the authors point of view, there are two main reasons for this. First, commercial software providing topology optimization solutions (e.g. CSA/NASTRAN, Altair OptiStruct, Quint Optishape and MSC Construct) are expensive to buy for small companies and require lengthy education of the operators. Second, due to the youth of the method, topology optimization is not mentioned even in modern text-books on optimization, and is seldom found in regular undergraduate and graduate courses. As a result, many students and engineers do not know about the promising method.

The present project is our attempt to help in improving the situation. We present a Web-site including a Web interface for a program performing topology optimization of statically loaded mechanical structures. The Web interface provides a simple way to try out the method, serves as a starting point for learning more about it and provides engineers and students with an easy-to-use software that can be used to develop insight into optimal structural design.

There are other examples of topology optimization software accessible throughout the Internet, for instance a down-loadable program based on evolutionary method (`www.ae.su.oz.au/wwwdocs/eso1.html`). However, in order to use them one has to down-load and install the software, the user is constrained by a computer platform and a speed of the computer. The technique, described in this paper, helps to avoid these deficiencies: all the user needs is a web browser.

Referring to the above-mentioned arguments, the main goals of the project are as follows.

1. To spread the concept and ideas of topology optimization among designers in various fields of engineering.
2. To use the interface as a computer-aided learning tool for students having courses on civil and mechanical engineering design, architecture, etc.

Our topology optimization program is based on mathematical programming techniques. Recently, many alter-

Received September 29, 2000
This manuscript was originally submitted and accepted for publication by the former journal Design Optimization on January 6, 2000

D. Tcherniak[1] and O. Sigmund[2]

Department of Mechanical Engineering, Solid Mechanics, Technical University of Denmark, DK-2800 Lyngby, Denmark
e-mail: `sigmund@fam.dtu.dk`

[1] *Current address:* Brüel & Kjær, Skodsborgvej 307, DK-2850 Nærum, Denmark
[2] *Corresponding author*

native procedures (e.g. genetic algorithms, hard-kill/soft-kill methods and evolutionary based algorithms) have been proposed for the solving of topology optimization problems. A third goal is to provide an easily accessible base for the comparison of results, convergence and speed of different methods.

The World Wide Web fits the requirements of the project very well. Among other well- known advantages such as accessibility, the World Wide Web offers many new informational technologies and distributed computing is among them. The distributed computing concept is a popular topic of discussion among users in the computer industry (Cheng *et al.* 1998; Isreb *et al.* 1997; Wagner and Castanotto 1997). This concept simply means that instead of having all the computation taking place on a user's local machine, the user's computer sends data to a powerful remote server, which returns results that are then displayed on the local machine. In our case this means that a client, even in a platform with relatively low computing ability, can run topology optimization problems without any special software installed but a Web browser. Thus a Web interface is a software which serves as a bridge between the client's computer and the remote server. Among other advantages, distributed computing solves the problem of program upgrading (one has only one copy of the program running on a server) but other questions like security and charging users (for commercial versions of programs) will rise.

The authors do not claim that the present paper contributes either to the scientific fields of topology optimization or to Web-based programming: a basic topology optimization formulation (cf. next section) and standard and well-documented Web tools were used for the interface. Thus the three goals of the paper, as we see them, are: (i) to report the existence of the site to engineers, designers and teachers who want to try topology optimization; (ii) to discuss educational aspects of the Web-based program; and (iii) to share our experience among those who want to apply web-based programming to their own fields of research.

The paper is built up as follows. Section 2 is a brief introduction to the topology optimization problem solved by the program. In Sect. 3 the educational aspects of the Web-based program are discussed and some statistics are presented. Section 4 describes the interface and also contains technical details of our implementation of the Web-based topology optimization program.

We invite the readers to test our program by visiting our Web Site: `http://www.topopt.dtu.dk`.

2 Topology optimization overview

As mentioned in the introduction, the topology optimization method has recently been applied to such various problems as mechanism design (Larsen *et al.* 1997; Sigmund 1997), material design and MEMS design. A Web page for the design of compliant mechanisms is available at our web site but here, we shall only discuss the most basic topology optimization problem consisting in the compliance minimization of static structures, considered in the present version.

In this version, the topology optimization method solves the problem of distributing a given amount of material in a design domain subject to load and support conditions, such that the stiffness of the structure is maximized. A simple example is shown in Fig. 1. The light-grey area denotes the design domain which is supported to at least exclude rigid-body motions and loaded by one or more forces applied in up to three different load-cases. Some areas of the design domain can be specified to consist of solid material (black areas) and some areas may be specified to be void (white areas).

Using the so-called power-law approach to topology optimization (Bendsøe 1989; Zhou and Rozvany 1991, e.g.), the design domain is discretized by N finite elements and the relative density x_e of material in each element is a design variable. The Young's modulus of material in each element is proportional to the element density raised to a power p, e.g. $E(x_e) = x_e^p \; E_0$. The power-law approach has previously been called the fictitious material approach since it was believed that an isotropic material with Young's modulus described by above power-law was non-existent. However, a recent paper by Bendsøe and Sigmund (1999) proves that the power-law approach is perfectly valid when the power satisfies a simple in-

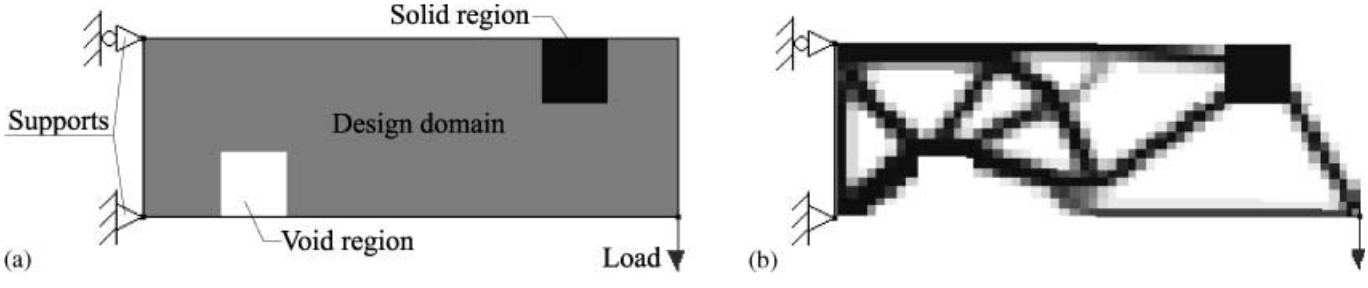

Fig. 1 (a) A typical topology optimization problem: a design domain with passive areas (white is fixed to be void and black is filled by a material), supports and a load. (b) An optimal design for the described problem

equality (e.g. p larger than three for Poisson's ratio equal to a third).

In order to prevent degenerated solutions, the program can handle multiple load cases but for simplicity, the number of load cases is limited to three equally weighted cases. The force vectors with dimensions equal to the number of degrees of freedom in the finite element problem are denoted $\mathbf{F}_1$, $\mathbf{F}_2$ and $\mathbf{F}_3$, respectively, and the corresponding displacement solutions are denoted $\mathbf{U}_1$, $\mathbf{U}_2$ and $\mathbf{U}_3$. The global finite element stiffness matrix is the same for all three load cases and is denoted $\mathbf{K}$.

With the above definitions, the topology optimization problem solved by our web-program can be written as

$$\left.\begin{array}{lll} \min\limits_{\mathbf{x}}: & \mathbf{F}_1^T\mathbf{U}_1(\mathbf{x})+\mathbf{F}_2^T\mathbf{U}_2(\mathbf{x})+\mathbf{F}_3^T\mathbf{U}_3(\mathbf{x}) & \\ \text{s.t.}: & V(\mathbf{x})/V_0 = f & \\ : & \mathbf{K}(\mathbf{x})\mathbf{U}_i(\mathbf{x}) = \mathbf{F}_i, \;\; i=1,2,3 & \\ : & \mathbf{0} < \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{1} & \end{array}\right\}, \qquad (1)$$

where V_0 is the volume of the total design domain and f is the fraction of the total volume that can be filled with material.

The optimal topology is independent of the Young's modulus, whereas the Poisson's ratio affects the optimal design slightly (more for high volume fractions) and also the choice of penalization power may cause the algorithm to converge to other (local) minima. However, in order to keep the user's input to the web program to a minimum, the Young's modulus, Poisson's ratio and penalization power are chosen to be constants ($E=1$, $v=1/3$ and $p=3$, respectively). Furthermore all applied forces have the same magnitude ($=1$) but the user can apply larger forces by placing several force vectors on top of each other.

The optimization problem (Fig. 1) is solved by a mathematical programming method (the Method of Moving Asymptotes (MMA) by Svanberg (1987) but for the simple compliance type objective function with only one linear constraint, an optimality criteria type algorithm could have been used as well. Using the MMA-algorithm, the optimization problem is solved iteratively involving alternating finite element analyses, sensitivity analyses, and material redistribution based on the solving of linearized subproblems. In order to ensure mesh-independent and checkerboard free designs we apply a filtering algorithm suggested by Sigmund (1994, 1997) (see also Sigmund and Petersson 1998 for an overview of numerical problems in topology optimization). The output of the topology optimization program is a bitmap picture of the optimal design (Fig. 1b). For more details on the topology optimization method, the reader is referred to the book by Bendsøe (1995), to the vast literature on topology optimization or to the other references by Sigmund.

3
Educational aspects of a web-based program interface

3.1
Requirements to meet

Since one of the main goals of the program is to use it in an educational/training process, let us consider the requirements that the web-interface should meet from this point of view.

First we want to allow students fast testing of many different sets of load/support cases. During a session, the student can try several (maybe 10 to 15) various combinations of load/support cases and volume fractions, in order to analyse and compare solutions that the program produces. This kind of "case study" can be very useful for the understanding of topology optimization. However, this calls for two requirements: (i) the program must be fast enough to solve problems in reasonable time (no more then 1–2 minutes); and (ii) since the user input is 2D geometrical information (design domain configuration, location of loads and supports), a graphical user interface is desirable. Menu and drag-and-drop features must allow the user to change the problem formulation in a fast, easy and intuitive way.

Then, the output of the target program (here, the topology optimization program) is a series of images. They show an animation of the optimization process, i.e. how the topology changes in order to accommodate the applied loads or in other words: the evolution of the optimal topology. We are sure that this animation is useful and instructive for understanding the topology optimization concept. It might be very practical if a user can watch this evolution although the transmission of graphical data via the Internet may be slow.

Then, some important details concerning the concepts of finite element analysis and topology optimization might effectively be demonstrated. One example is that the number of supports must be large enough to prevent rigid body motions, another example is that using symmetry, a more detailed design can be obtained with the same number of elements and computational time. Likewise a lecturer can explain the importance of using several load cases – an important concept of topology optimization which, if ignored, may lead to seriously degenerated designs.

Finally, it could be useful to provide several "ready-to-submit" examples of problem formulations in order to help the user in getting acquainted with the program and its possibilities.

Besides this, we took the following considerations into account in the layout of the interface.

1. The program should run in any computing platform under a web-browser (at least under the new enough versions of the two most popular ones – Netscape Navigator/Communicator and Microsoft Internet Explorer).

2. The user interface should be attractive.
3. It should be self-explanatory with only a brief on-line documentation (Internet surfers rarely read manuals or even short accompanying explanations).
4. The number of controls should be minimized: we allow our user to change only the few most important problem parameters.
5. The checking part of the user's interface should not be very restrictive and annoying: it should not forbid but inform and recommend.
6. The data transfer should be minimized: the realization of the client's part should be compact to keep the down-loading time reasonable.
7. The server part should be able to process the problems of several users simultaneously with only a reasonable speed loss.
8. It has to be robust and tolerant to user's incorrect inputs.
9. For users interested in theory there should be links for detailed information.

3.2 Appearance of the interface

The interface for distributed computing programs normally consists of two parts: a client's part and a server part. The client's part provides a user's interface, checks the user's input, sends data to the server and finally displays server output on the client's computer. The server part is hidden from the user, it registers the user, reads the data, starts a target program (which provides the topology optimization itself) and then supplies the client with output data generated by the target program.

We used a Java applet technology (Tyma *et al.* 1996) for the implementation of the client's part. An applet is a small Java program that is down-loaded via the Internet and runs on a client machine. Using Java one is able to produce safe, functional, platform-independent, attractive and fairly fast programs or interfaces. Some technical details of the implementation can be found in the next section.

Our applet is a part of the Topology Optimization Site (`http://www.topopt.dtu.dk`). When down-loaded, it presents a graphics screen as many ordinary graphics editors do. By means of the menu and a drag-and-drop mechanism, the user defines a design domain and volume fraction, applies loads and supports. We tried to work out the graphics interface as simple as possible; the resulting problem formulation looks like a drawing in a standard text-book on mechanics.

When down-loaded, the applet appears with an example problem formulation. This example can directly be submitted or it can be modified or reset.

Pressing the "submit" button, the user first initiates the checking procedure. We put efforts into making this procedure less restrictive. Only few user's actions might stop the data submission and require to change the input, e.g. "Error: No forces applied." or "Error: Supports allow rigid body motion! Please add supports". Most messages just give information to the user, e.g. "Warning: The force is applied outside the domain. The force is ignored". Of course, it is infeasible to predict and catch all possible incorrect inputs at the earliest stage of the data submission. However, the target program is stable enough to prevent a hang-up due to an incorrect input.

After the submission, the target program (located in a remote server) starts to solve a user problem. The process is iterative, results after every iteration are transmitted to the users computer and displayed in a special window. Provided the problem is not very complicated and the network connection is fast enough, the series of resulting images present a reasonably smooth animation which illustrates the optimization process. The user can interrupt the process and modify the input. We decided to keep windows with old designs; thus making it possible to compare results of different inputs which is certainly useful for educational reasons. The common users session is shown in Fig. 2.

The applet is accompanied by links, where we explain the details of the user's interface and some basic ideas of topology optimization (e.g. what a design domain is and what the volume fraction means, why multiple load cases should be used, etc.). There are also links to examples of real structures such as a bridge, a wheel, etc., and links to some useful hints and tips, showing for example, how to simplify a problem using symmetry, etc.

As was mentioned above, the program is a part of a Topology Optimization site which also includes an introduction to topology optimization theory and overview of techniques used in the method. Thus, we hope, the site can be a good starting point for those who want to learn more about topology optimization methods.

3.3 Statistics and examples of users designs

The pilot project was first made publicly available in December 1998. The present version started in February 1999. The study of log-files allows one to make detailed studies of users habits. At the time of writing (January 2000), the program was tried by 400 visitors (distinguished by their IP addresses) and has been started 3760[1] times (on average, about 10 times per day ranging from few in February to up to 280 per day when it was being used in student courses). As can be seen, many users run the program several times, the number on an average is about 9 times, however, it exceeds 100 times for some visitors. In total, the program performed about 232 000 iterations and this took approximately 131 000 seconds (about 37 hours) of running time (measured from the start of the target program until the last requested picture has been

[1] the corresponding numbers are 1087 and 10 993 at the time of final proof reading (April 2001)

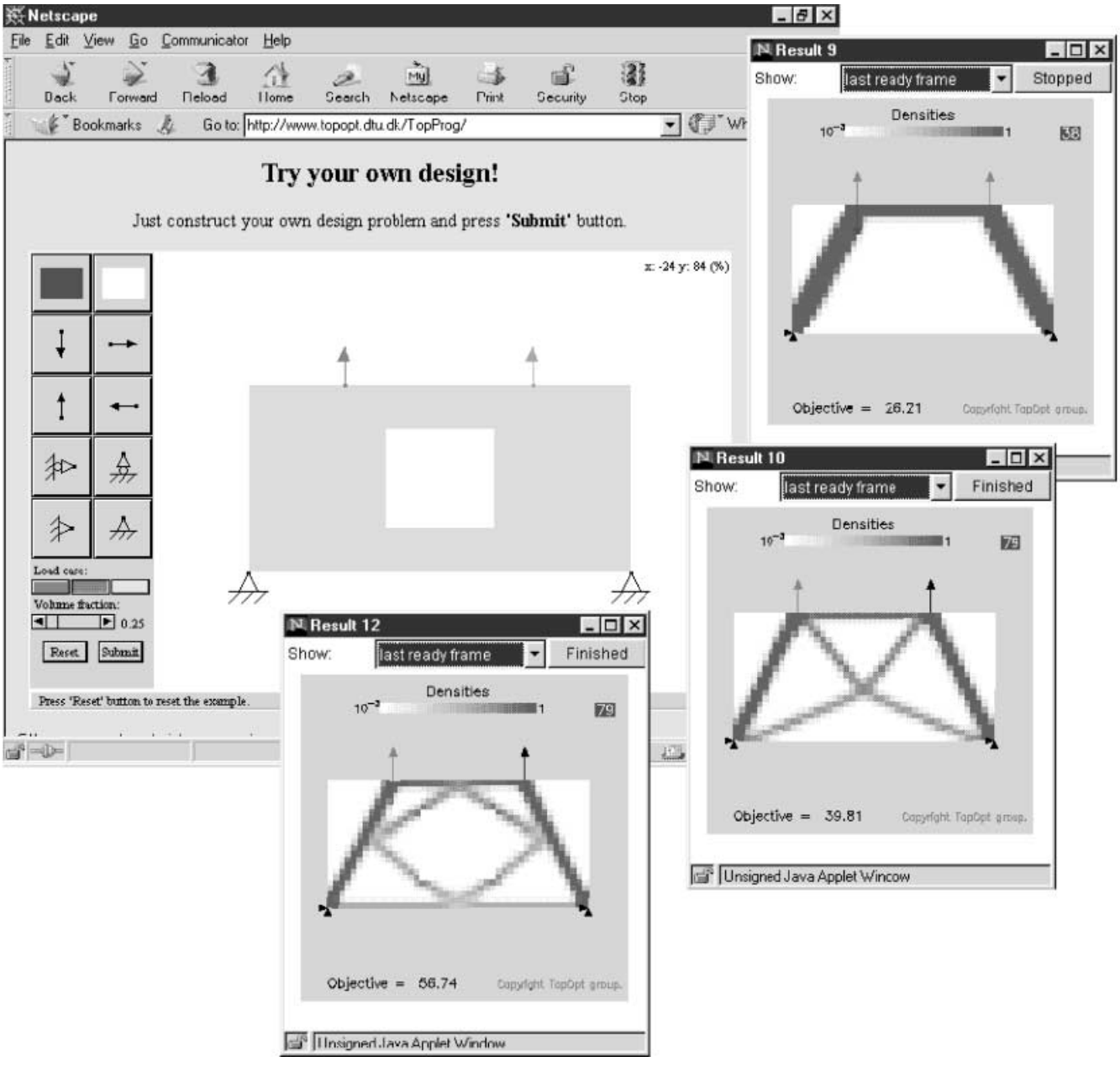

Fig. 2 A generic user's session. The three solutions correspond to a problem formulation which has been modified during a design session. The user got the design for only one load case (right window), then reformulated the problem using two load cases (middle window), and finally added a void passive area in the middle of the design domain (the problem formulation actually shown in the applet) to get the design in the bottom window

down-loaded by the user). On average one iteration was performed in 0.56 seconds, however this time is varying from 0.5 second to 90 seconds mostly due to very different speed of network connection. The average number of iterations per one program run is 54 (maximum allowed iterations is 80) which means that many users interrupt the process before it converges.

As expected, the help pages of the site, where we tried to highlight some ideas of topology optimization and not obvious details of the user interface, were not very popular – on average they were visited about 100 times. This confirms the idea that the user interface must be as simple as possible.

During the year of operation we stored final designs (maximum of five per user) of our visitors in a special directory (`http://www.topopt.dtu.dk/TopProg/Gallery/Galery1.html`); some of which are shown in Fig. 3. Surprisingly this directory gained a lot of popularity among our users – this again confirms that this kind of "case study" can serve as a very instructive way of learning.

Other site statistics are also available at `http://www.topopt.dtu.dk/Statistics`.

Although we ask users to mail us comments or questions, only a few users have given us any response. The few comments we did receive, mainly expressed the users

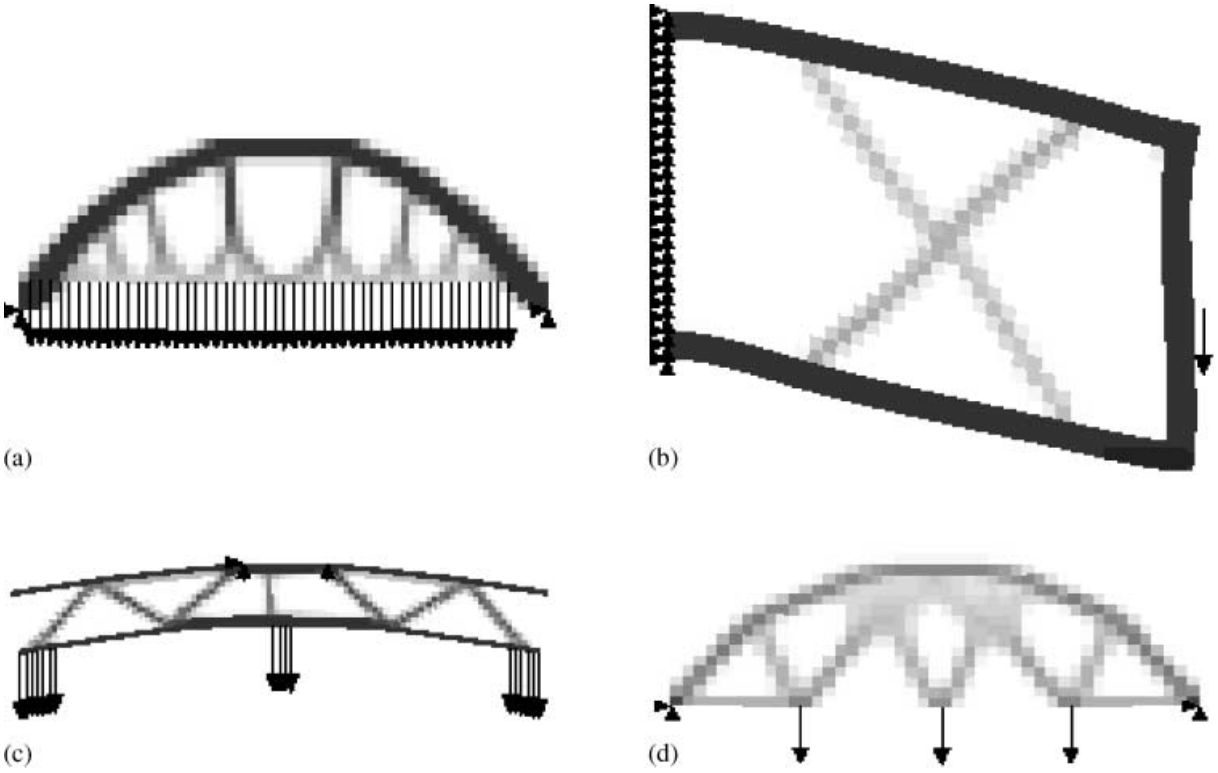

Fig. 3 Designs of our visitors: (a) By skiff1.univ-brest.fr (25.05.99); (b) by hamachi.engin.umich.edu (26.06.99); (c) by dial37.inet.zitech.dk (18.06.99); (d) by dhcp105.me.psu.edu (27.07.99) (the forces belong to three different load cases). The triangles denote supports

surprise at how fast the program works (c.f. next section for a partial answer to this question). We also received positive oral feedback from users in academia who have used it in their courses and from a Danish company that used the program to get ideas for a new engine part.

4 Implementation of the interface

This section contains the description of the technologies on which the implementation of our program is based and some technical details of the implementation. The section is mostly addressed to those researchers who want to develop similar web-interfaces for their own programs in their research areas.

4.1 The target program

In our project we use a topology optimization program developed in our lab. It is a well-developed program which is fast and robust. The program is written in Fortran 77 language and it uses the PGPLOT graphics package to generate the output images in CompuServe's GIF file format.

In our project we use a version of the program that allows one to optimize statically loaded mechanical structures according to the problem formulation in Sect. 2. The software is developed in our lab and does not make use of any commercial or public finite element library. This way, it is possible to write a very efficient code that is optimized for the topology optimization problem at hand. For a remote client we set a limit on the number of program iterations to 80 and allow the client to use a mesh with up to 1000 4-node linear finite elements. Experience shows that this resolution is fine enough for test and educational purposes and modest enough to keep the calculation time in the wanted range. In this case, our server (based on the Intel Pentium-II 450 MHz processor with 256 MB of RAM running the Linux operating system) makes approximately 100 iterations per minute (for a typical problem and if only one copy of the target program is running at the same time). Thus it takes about 50 seconds to produce the whole series of images. The size of one GIF file is approximately 3 Kbytes (GIF87, 256×256 pixels), so we have about 240 Kbytes to be transmitted to the user.

4.2 On image transfer technologies

The client-server interaction is the most important part of any interface. In our case it has to meet two main criteria.

1. The interface should deliver an image as soon as it has been generated by the target program. It is desirable to provide the user with a smooth animation if this is allowed by the complexity of the user's problem, connection quality and server load.
2. The server should terminate the target program and clean up the image files, which were not called for, if the user has interrupted the calculations, left a browser or jumped to another web-page.

There is a standard protocol that meets these requirements – the so-called Netscape multipart/x-mixed replace push-server. Based on this protocol, a browser gets data by portions and successively puts them into a screen: when the next portion is coming, the browser replaces the old content by the new one. A user can terminate the transfer, for example by pressing the browsers stop button, this will also stop the target program. The push-server method seems to be robust and could easily be implemented due to its simplicity. However, this protocol has a disadvantage: only few browsers support it (e.g. Netscape Navigator or Communicator), the Microsoft Internet Explorer does not. Working on an Internet-based program, we wanted the program to operate with most browsers, which is why we rejected this protocol for the public version of the project. (It was used in a pilot realization of the project; one can test it at `http://www.topopt.dtu.dk/Push`).

Another option for making web-animations is the so-called pop-server method. In this case a web-browser continuously reloads a web-page content in a specified time whether or not the content actually has been changed. On the server side, a target program has to replace the old web-page content (or its component, e.g. an image) by the new one. The protocol is based on the "refresh" feature of the HTML and thus must be recognized by all modern web-browsers. The obvious disadvantage of this protocol is that it generates enormous and unnecessary network traffic. Also it should be noted that there is no straightforward way to stop the target program when a client breaks the connection.

4.3 The protocol implementation

Due to the disadvantages of the above-mentioned methods, we decided to work out a custom protocol which is organized as follows. As the target program has prepared a GIF file, the server connects to the client and sends the name of the file as a signal that it is ready. Until this moment the client is waiting for this signal. Then it reads the file name and checks whether it should show the image or omit it. The latter can happen if the user ordered to draw not each but, say, every fifth image. If the image should be drawn, the client program starts to down-load it and renders it to a screen when the down-loading is completed. The described scheme is simple and meets both criteria outlined above. It reduces the network traffic, because only short file names are transferred via Internet and only requested files are down-loaded. The client and server keep the connection open, thus if the user breaks the connection, the server will know about this and react accordingly: it terminates the target program and makes a clean up.

There are two bottlenecks in the way we supply our user with the resulting images. The first one is the solving of the user's topology optimization problem on the server. The time mainly depends on the bandwidth of the finite element problem and the present CPU load. The second bottleneck is the delivery of the images to the user, and this depends on the connection quality. We do not know beforehand which one will be the actual bottleneck in every specific case. That is why it is desirable to separate the process of solving the problem from the process of the image delivery and make these processes asynchronous. This explains our decision to implement the above-mentioned protocol by means of Java's thread technology. A thread is a lightweight process which runs simultaneously with other processes. Java defines threads as an integral part of the language and includes a number of features that allow one to easily write Java programs that create multiple threads.

Figure 4 shows the protocol implementation. After the "Submit" button is pressed, the applet starts two threads. The first one is responsible for a connection with the server: it requests the connection, sends the problem description and "talks" with the server. The second thread is responsible for the display of incoming images: it opens a new window, down-loads image files and draws them on the screen. The two threads are asynchronous. They share one common resource – the list of the names of ready image files: as soon as the server connection thread reports the file is ready, the client connection thread appends the file name to the end of the list. The window thread successively down-loads the files named in the list. It can down-load each file from the list or every 2nd, 5th or 10th or last ready file. This depends on the user's choice and supposed to be based on his/her estimation of the network connection speed.

The server is written in Java. In order to get more flexibility, we used low-level network communication by means of sockets, which are provided by the standard Java's API. The main program is an infinite loop, listening to a specified port. As a connection is requested, the code checks if the current number of users does not exceed the allowed number (in our case three simultaneous connections are allowed) and, if it does not, registers the user. Then the program spawns the server connection thread which is responsible for the serving of the client. The thread answers the client, reads the client's data and starts the target program with this data. As the target program has generated a new GIF file, the thread sends its name to the client. The thread does not transmit the files: a standard HTTP daemon is in charge of this operation.

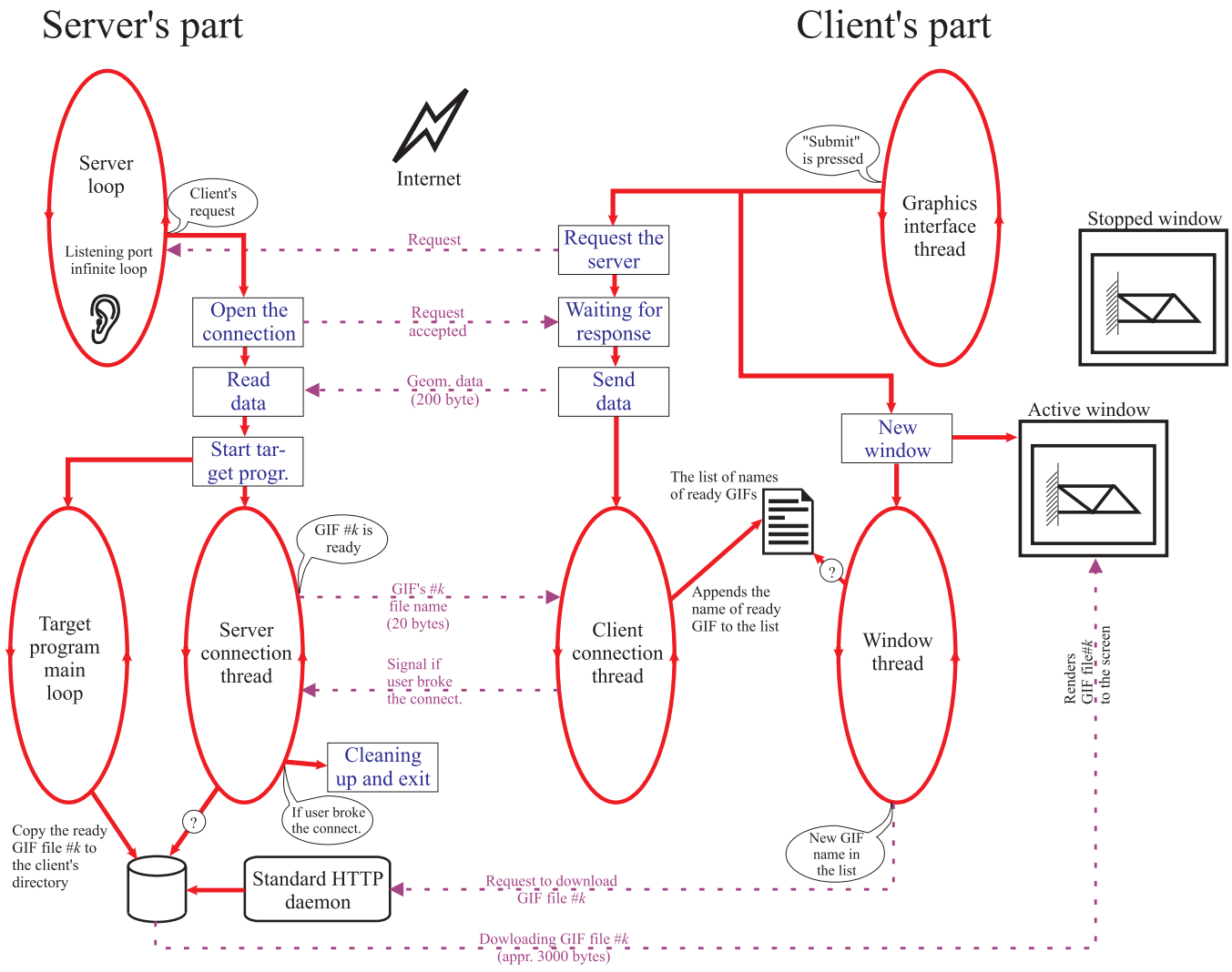

Fig. 4 Connection flow and interaction of threads. Oval here denotes the thread's run- loops, solid lines arrow – operation flow, dashed arrows – transmission via the Internet, – the "event" happened, – a request

If the user ends the connection, the client connection thread notifies the server. The server thread terminates the target program and makes a clean-up. If something irregular happens, this situation is processed by Java's catch-exception mechanism and in any case the server fulfills the prescribed cleaning operations.

When the program became publicly available, we experienced the following problem: sometimes the communication was broken when the server thread was waiting for client's data. In this case the situation cannot be processed by Java's exception mechanism; and the server waits forever. (This case is explained in detail by Cassady-Dorion *et al.* 1997, p. 388). This results in "hang-up" of the server communication thread. If the situation repeats, the number of active threads achieves the limit; after which the server rejects all other incoming requests. The only cure was a manual reboot of the server program. To prevent this, a simple time-out mechanism was introduced: if the thread remains active longer then it is allowed (about 5 minutes in our case), the server terminates it. As soon as this mechanism was implemented, the server became significantly stable.

Based on our experience we shall note that the use of Java allowed us to significantly shorten the period of program development. The above-mentioned features of Java such as threads, exception mechanism and in-built network programming tools allow a rapid development of Internet-wrappers for existing code regardless of the language in which it was written or the computing platform on which it is running.

5 Summary and conclusions

In this paper, a web-based interface to a topology optimization program is described. We believe that the interface will assist in promoting the topology optimization approach as a modern and useful design tool suitable for various fields of engineering. Furthermore, the interface can be used as a computer-aided learning tool and it can

be considered as an example of the application of distributed computing concepts.

Obviously, much remains to be done. It is desirable to implement the same interface to other problems where the topology optimization approach is beneficial, e.g. to the design of compliant mechanisms, smart materials or MEMS. There are some technical issues which need to be further tackled such as data flow rate over the network, compatibility with other web-browsers, etc. A mechanism of charging users should be developed for a commercial version of the program.

Acknowledgements The authors gratefully acknowledge the support by the Danish Technical Research Council through the THOR/Talent-programme: Design of MicroElectroMechanical Systems (MEMS). The authors would also like to thank Krister Svanberg from KTH, Sweden for supplying the MMA-subroutines.

References

Bendsøe, M.P. 1989: Optimal shape design as a material distribution problem. *Struct. Optim.* **1**, 193–202

Bendsøe, M.P. 1995: *Optimization of structural topology, shape and material*. Berlin, Heidelberg, New York: Springer

Bendsøe, M.P.; Kikuchi, N. 1988: Generating optimal topologies in optimal design using a homogenization method. *Comp. Meth. Appl. Mech.Engrg.* **71**, 197–224

Bendsøe, M.P. and Sigmund, O.: 1999, Material interpolations in topology optimization, *Archive of Applied Mechanics* **69**, 635–654

Cassady-Dorion, L.; Brumbaugh, M.; Maheshwari, S.; Wright, J.; Brookshier, D.; Last, B.; Mathis, J. 1997: *Industrial Strength Java*. Indianapolis, IN: New Riders Publishing

Cheng, K.; Harrison, D.K.; Pan, P. 1998: Implementation of agile manufacturing – an AI and internet based approach. *J. Mat. Process. Tech.* **76**, 96–101

Hinton, E.; Toropov, V.V. 1999: Editorial on education and training. *Design Optimization* **3**

Isreb, M.; Khan, A.I.; Parker, B.A. 1997: Adaptive finite element mesh refinement – a CAL module on the www. *Comp. Struct.* **65**, 169–175

Larsen, U.D.; Sigmund, O.; Bouwstra, S. 1997: Design and fabrication of compliant mechanisms and material structures with negative Poisson's ratio. *J. Microelectromech. Sys.* **6**, 99–106

Sigmund, O. 1994: *Design of material structures using topology optimization*. PhD thesis, Department of Solid Mechanics, Technical University of Denmark

Sigmund, O. 1997: On the design of compliant mechanisms using topology optimization. *Mech. Struct. Mach.* **25**, 495–526

Sigmund, O.; Petersson, J. 1998: Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Struct. Optim.* **16**, 68–75

Svanberg, K. 1987: The method of moving asymptotes – a new method for structural optimization. *Int. J. Numer. Meth. Eng.* **24**, 359–373

Tyma, P.M.; Torok, G.; Downing, T. 1996: *Java Primer Plus*, Corte Madera, CA: Waite Group Press

Wagner, R.; Castanotto, G. 1997: Fixturenet: Interactive computer-aided design via the world wide web. *Int. J. Human-Computer Stud.* **46**, 773–788

Zhou, M.; Rozvany, G.I.N. 1991: The COC algorithm. Part II: topological, geometry and generalized shape optimization. *Comp. Meth. Appl. Mech. Eng.* **89**, 197–224