Symbolic Methods in Numerical Optimization

Giuseppe Cerbone
Thomas G. Dietterich
Oregon State University
Computer Science Department
Corvallis, OR 97331-3202

91-30-7

Symbolic Methods in Numerical Optimization

Giuseppe Cerbone [†]
cerbone@cs.orst.edu
Thomas G. Dietterich
tgd@cs.orst.edu

Oregon State University
Computer Science Dept.
Corvallis, OR 97331-3202
Phone (503) 737-3273

Abstract

Many important application problems can be formalized as constrained non-linear optimization tasks. However, numerical methods for solving such problems are brittle and do not scale well. Furthermore, for large classes of engineering problems, the objective function cannot be converted into a differentiable closed form. This prevents the application of efficient gradient optimization methods—only slower, non-gradient methods can be applied. This paper describes a method to speedup and increase the reliability of numerical optimization by (a) optimizing the computation of the objective function, and (b) splitting the objective function into special cases that possess differentiable closed forms. This allows us to replace a single inefficient non-gradient-based optimization by a set of efficient numerical gradient-directed optimizations that can be performed in parallel. In the domain of 2-dimensional structural design, this procedure yields a 95% speedup over traditional optimization methods and decreases the dependence of the numerical methods on having a good starting point.

Category: Machine learning, speedup learning, optimal design.

[†]This research was supported by NASA Ames Research Center under Grant Number NAG 2-630.

1 Introduction

Many important applications can be formalized as constrained optimization tasks. For example, we are studying the engineering domain of two-dimensional (2-D) structural design. In this task, the goal is to design a structure of minimum weight that bears a set of loads.

Figure 1 shows a solution to a design problem in which there is a single load (L) and two stationary support points (S1 and S2). The solution consists of four members, E1, E2, E3, and E4 that connect the load to the support points. In principle, optimal solutions to problems of this kind can be found by numerical optimization techniques. However, in practice [Van84] these techniques are very slow, and the solutions obtained depend on the choice of starting points. Hence, their applicability to real-world problems is severely restricted.

To overcome these limitations, we propose to augment numerical optimization by first performing a symbolic compilation stage to produce objective functions that are faster to evaluate and that depend less on the choice of the starting point. These goals are accomplished by successive specializations of the objective function that, in the end, reduce it to a collection of independent functions that are fast to evaluate, that can be differentiated symbolically, and that represent smaller regions of the overall search space.

Our optimization schema differs from techniques currently used in the machine learning community. Our approach relies on the specialization of the problem via incorporation of constraints prior to optimization. Braudaway [Bra88] designed a system along the same principle. However, to our knowledge, very little work has been done in using symbolic techniques and domain knowledge to speedup numerical optimization tasks. In contrast, the current trend in the machine learning community focuses on methods, such as Explanation Based Learning (EBL) [Ell89], capable of generating rules. Minton [Min88] shows how these methods can result in a slow-down of the overall process. In addition, EBL methods have had little success in the task of optimizing numerical procedures. We conjecture that one of the reasons is the dependence of EBL methods on the trace of the problem solver. The trace of a numerical optimizer gives little information on the structure of the problem. Therefore, in mathematical domains, EBL-derived rules are too detailed to produce any appreciable speedup.

The remainder of the paper is organized as follows. Section 2 presents the

2-D structural design task. This is followed in Section 3 by an overview of numerical optimization methods, their limitations, and the proposed solution, which is illustrated using a simple example. The proposed method is then applied to a collection of randomly generated examples and the results of these experiments are reported in Section 4. The experiments show that, for a certain family of problems, the compilation stage produces a substantial improvement in the performance of the optimization methods. Benefits and limitations of the method are summarized in Section 5, which also outlines future work.

2 Task description

Table 1 describes the 2-dimensional structural design task that we are attacking. Figure 1 shows an example problem in which L is the load and S1 and S2 are two supports. The so-called "topology" is given as a graph structure containing four edges (the members) and four vertices (the load, the two supports, and an intermediate connection point C). The topology does not specify the lengths of the members or the location of C.

The goal of the design process in this example is to find a location for C that minimizes the weight of the structure. The position shown in the figure gives the minimum-weight solution. In this solution, members E1 and E3 are in tension (they are being "stretched"), while members E2 and E4 are in compression. Tension members will be referred to as "rods" and indicated by thin lines. Compression members will be referred to as "columns" and indicated by thick lines.

The task shown in Table 1 is actually only one step in the larger problem of designing good structures. In general, structural design proceeds in three steps [PS70, Van84]. First, the problem solver chooses the topology, which specifies the locations of the loads and supports and the connectivity of the members. Then, the second step is to determine the locations of the connection points (and hence the lengths, locations, internal forces, and cross-sectional areas of the members) so as to minimize the weight of the structure. This is usually accomplished by numerical non-linear optimization techniques, and it is the focus of this paper. The third and final step in the process optimizes the shapes of the individual members. This can often be accomplished by linear programming.

In addition to focusing only on the second step, we have introduced several

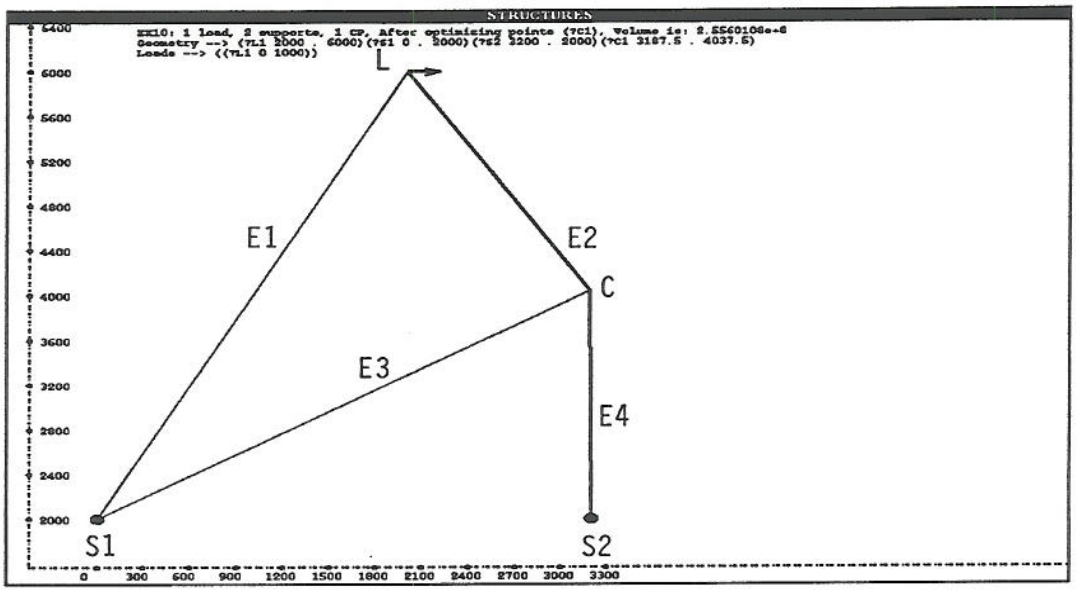

Figure 1: A solution to a 2-D structural design problem with given topology.

Table 1: The 2-D Design Task.

Given:	A 2-dimensional region R A set of stable points (supports) A set of external loads with application points within R A topology specifying the number of members and their interconnection graph.
Find:	The positions of all intermediate connection points such that the structure has minimum weight and is stable with respect to all external loads.

simplifying assumptions to provide a tractable testbed for developing and testing knowledge compilation methods. Specifically, we assume that structural members are joined by frictionless pins, only statically determinate structures[1] are considered, the cross section of a column is square, columns and rods of any length and cross sectional area are available, and supports have no freedom of movement.

Given these assumptions, the weight of a candidate solution is usually calculated by a three-step process. The first step is to apply the *method of joints* [WS84] to determine the forces operating in each member. Once this is known, the second step is to classify each member as compressive or tensile, which is important, because compressive and tensile members have different densities. The third step is to determine the cross-sectional area of each member. The load that a member can bear is assumed to be linearly proportional to its cross-sectional area. Finally, the weight of each member can be computed as the product of the density of the appropriate material, the length of the member, and the cross-sectional area of the member.

The last two steps can be collapsed into a single parameter k: the ratio of the density per-unit-of-force-borne for compressive members to density per-unit-of-force-borne for tensile members. With this simplification, instead of minimizing the weight, we can minimize the following quantity

$$V = \sum_{\substack{\text{tensile} \\ \text{members}}} \|F_i\| \, l_i + \; k \sum_{\substack{\text{compressive} \\ \text{members}}} \|F_i\| \, l_i,$$

where F_i is the force in member i, and l_i is the length of member i. This is the initial objective function for the work described in this paper.

We conclude this section with a brief description of the method of joints, which is one of the methods used to calculate the F_i in statically determinate structures. The method of joints computes these forces by solving a system of linear equations as illustrated, for the problem in Figure 1, in Table 2. The matrix of coefficients is called [WS84] the *axial* (or *static*) matrix and the vector of givens is defined as the *load vector*. In Figure 1, let $\mathsf{L} = (x_l, y_l)$, $\mathsf{C} = (x, y)$, $\mathsf{S1} = (x_1, y_1)$, and $\mathsf{S2} = (x_2, y_2)$, be the cartesian coordinates of the load, the connection point, and the two supports, respectively. In addition, let p and γ be the magnitude and direction of the load. The internal forces in each member are obtained by first constructing the axial matrix and load vector and then

[1] A statically determinate structure contains no redundant members, and hence, the geometrical layout of the structure completely determines the forces acting in each member.

Table 2: Method of Joints for the example in Figure 1.

$$\begin{pmatrix} cos(\alpha_1) & cos(\alpha_2) & 0 & 0 \\ sin(\alpha_1) & sin(\alpha_2) & 0 & 0 \\ 0 & cos(\alpha_2+180) & cos(\alpha_3) & cos(\alpha_4) \\ 0 & sin(\alpha_2+180) & sin(\alpha_3) & sin(\alpha_4) \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix} = \begin{pmatrix} Lcos(\gamma) \\ Lsin(\gamma) \\ 0 \\ 0 \end{pmatrix}$$

where:

$$\begin{aligned} cos(\alpha_1) &= (x_1 - x_l)/l_1, & cos(\alpha_2) &= (x - x_l)/l_2 \\ cos(\alpha_3) &= (x_1 - x)/l_3, & cos(\alpha_4) &= (x_2 - x)/l_4 \\ sin(\alpha_1) &= (y_1 - y_l)/l_1, & sin(\alpha_2) &= (y - y_l)/l_2 \\ sin(\alpha_3) &= (y_1 - y)/l_3, & sin(\alpha_4) &= (y_2 - y)/l_4 \end{aligned}$$

and l_i's are Euclidean distances:

$$\begin{aligned} l_1 &= \sqrt{(x_1 - x_l)^2 + (y_1 - y_l)^2} \\ l_2 &= \sqrt{(x - x_l)^2 + (y - y_l)^2} \\ l_3 &= \sqrt{(x - x_1)^2 + (y - y_1)^2} \\ l_4 &= \sqrt{(x - x_2)^2 + (y - y_2)^2}. \end{aligned}$$

solving the system of equations for the unknown internal forces. Table 2 shows the symbolic system of equations for the example in Figure 1 with unknown forces F_1, F_2, F_3, and F_4 and with the coordinates of all the points explicitly substituted.

Now that we have defined the 2-dimensional design task and formulated it as a non-linear optimization problem, let us turn, in the next section, to a brief review of existing techniques for optimization and to the proposed method.

3 Optimization

Classical optimization textbooks [Van84, PW88] present a comprehensive survey of optimization methods and of various techniques for conducting the search for an optimal solution. The schema illustrated in Figure 2 is typical of many domain independent non-linear optimization methods. The process is iterative. Starting at some initial point, the objective function is evaluated and the termination criteria are tested. If the test fails, a new point is generated by taking a step, of some chosen length in some chosen direction, away from the current point. Each point defines a set of values for the independent variables in the objective function.

Most optimization algorithms differ primarily in the criteria used to choose the direction along which to optimize. Some optimization methods (e.g., Pow-

ell's method [Van84]) choose the direction and step size using only evaluations of the objective function. Other methods, such as gradient descent and its variations [PW88], require computation of the partial derivatives of the objective function to choose the new direction of optimization. Still other methods approximate the partial derivatives numerically by evaluating the objective function at many points.

The primary computational expense of numerical optimization methods is the repeated evaluation of the objective function. An advantage of gradient descent methods is that they need to evaluate the objective function less often, because they are able to take larger, and more effective steps. Of course, they incur the additional cost of repeatedly evaluating the partial derivatives of the objective function. Hence, they produce substantial savings only when the reduction in the number of function evaluations offsets the cost of evaluating the derivatives.

In engineering design, the objective function is typically very expensive to evaluate, since it reflects many of the specifications for the design problem. Furthermore, it is often the case that the objective function lacks a differentiable closed-form. For example, in our objective function from the previous section, the fact that the constant k is applied only to compressive members makes it impossible to obtain a differentiable closed-form. The signs of the internal forces must be computed before it is possible to determine which members are compressive.

Given that the speed of numerical optimization is determined by the cost and frequency of evaluating the objective function, there are two obvious ways to speed up the process: (a) reduce the cost of each evaluation of the objective function and (b) reduce the number of evaluations by finding a closed-form for the derivative of the objective function, so that gradient descent methods can be applied. In addition, after the gradient has been computed, it is possible to obtain a further speedup by computing, at compile time, its linear or quadratic Taylor approximation.

We have developed an approach that pursues these directions. The basic idea is to perform a "compilation" stage prior to numerical optimization. During compilation, the objective function is subjected to successive specialization phases to incorporate constraints into the function. First, the given topology is incorporated into the objective function. This allows us to compute symbolically the axial matrix and the load vector (see Section 2). We then apply symbolic procedures to solve and simplify the system of equations and obtain

a closed-form expression for the forces. In principle, an infinite number of topologies should be explored; however, in practice [Fri71], only a few of them need be considered.

The second specialization step is to plug in the givens of the problem and partially evaluate the resulting mixed symbolic/numeric expression. For our examples, the givens of the problems are the loads and supports; however, one may wish to analyze a structure subject to different inputs such as various loading conditions or support locations. In such cases it is possible to leave those values in symbolic form and substitute their numerical values at runtime.

The third compilation step is to split the objective function V into cases according to an abstraction called the *stress state*[2]. The *stress state* of a structure is a vector indicating, for each member, whether it is compressive or tensile.[3] The i-th element in the vector is $+1$ if the i-th member in the structure is tensile, and -1 if the i-th member is compressive. If there are m members in a structure, then there are 2^m possible stress states, although only a few of them are physically realizable. For the example in Figure 1, the stress state of the optimal solution is $(+1, -1, +1, -1)$.

When the objective function is specialized according to stress state, the result is a collection of special-case objective functions $\{V_1, \ldots, V_n\}$. Because each V_j corresponds to one stress state, it is possible to tell, at compile time, which forces should be multiplied by k. Hence, each V_j is differentiable, and this enables us to employ gradient-based optimization techniques that, typically, are faster than methods based only on evaluating the objective function alone.

To further speedup the optimization process, each special case can be solved in parallel on independent machines. All that is required is to compare the solutions found for each V_j and choose the best one. Figure 3 gives a schematic diagram of this "compiled optimization" method.

Let us follow an example through each of the three specialization phases.

Topological simplification. Given a topology, the elements of the axial matrix, the load vector, and the lengths can be specified symbolically as functions of the coordinates of givens and unknowns in the problem. Table 2 shows the algebraic relationship between givens and unknowns for the topology in Figure 1. The system of equations can be solved symbolically to produce a

[2]The authors wish to thank Dr. David G. Ullman for suggesting this term.

[3]This is a generalization of the standard definition (see [Gor78]) of load path.

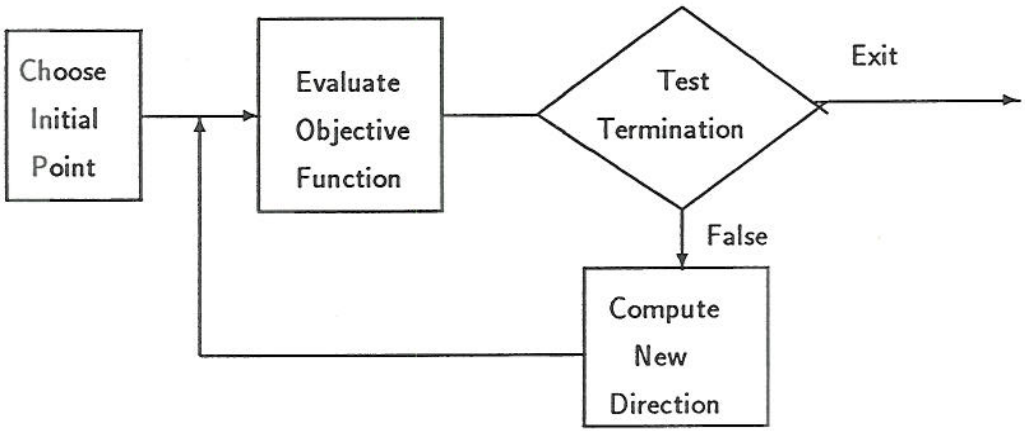

Figure 2: Traditional optimization schema.

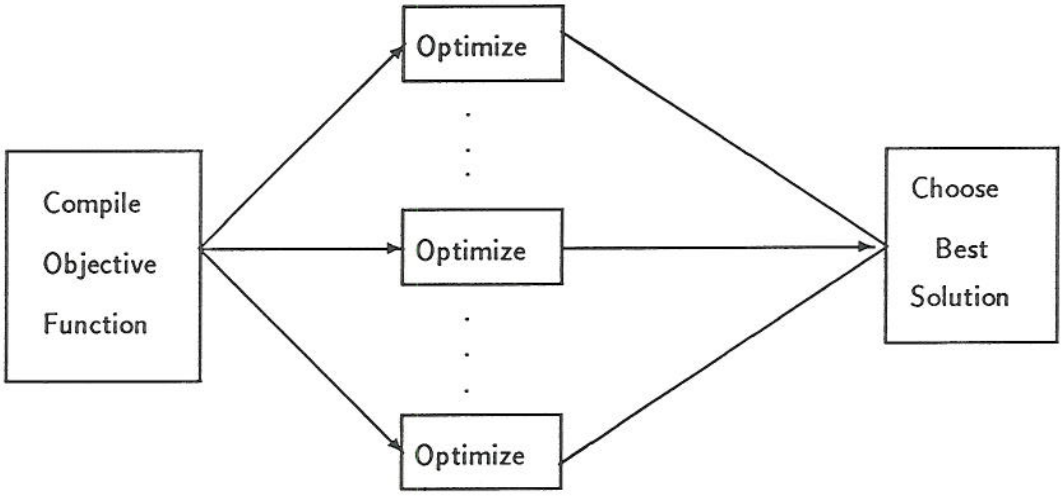

Figure 3: Optimization schema with compilation stage.

Table 3: Closed-form of the internal force for member E1 in Figure 1.

$$
\begin{aligned}
&\textit{Internal Force in member } \mathsf{E1} = \\
&\quad p\Big[+\big((x_1-x_l)^2+(y_1-y_l)^2\big)\\
&\qquad\qquad \big((x_2-x)(y_1-y)(y-y_l)-(x_1-x)(y_2-y)(y-y_l)\big)cos(\gamma)+\\
&\qquad\qquad \big((x_1-x)(x-x_l)(y-y_l)-(x_2-x)(x-x_l)(y_1-y)\big)sin(\gamma)\Big]\Big/\\
&\quad \Big[\big((x_2-x)(y_1-y)-(x_1-x)(y_2-y)\big)\big((x-x_l)(y_1-y_l)-(x_1-x_l)(y-y_l)\big)\Big]
\end{aligned}
$$

Table 4: Internal force for member E1 in Figure 1 with givens.

Internal Force in member E1 = $2236(y-6000)/(y-2x-2000)$

simplified closed-form expression for each of the internal forces in each member. Table 3 shows the expression of the force in the member E1 that joins the load L and the support S1.

Instance simplification. During the second phase, the expressions obtained with topological simplifications are partially evaluated with respect to the givens of the problem chosen by the user. For the example in Figure 1, we choose loads and supports as givens and the expression of the internal force in E1 is shown in Table 4 which shows that the force is now reduced to a closed-form expression of the coordinates x and y of the (unknown) connection point C.

Case analysis. For illustrative purposes, in Figure 4 we have plotted the volume of the structure for the topology in Figure 1 as a function of the coordinates x and y of the connection point C. Each unimodal region in the figure corresponds to one or more stress states; for instance, $(+1,-1,+1,-1)$ corresponds to region $R1$. This correspondence between stress states and unimodal regions is exploited by the case analysis, which partitions the whole region and produces one objective function per stress state. As an example, Table 5 shows the objective function for $(+1,-1,+1,-1)$. Each of these specialized objective functions is differentiable and, at compile time, it is also possible to compute its partial derivatives with respect to the optimization variables. The gradient can then be approximated via Taylor series expansion to produce a further speedup in the optimization process. The quadratic approximations

Table 5: Partially evaluated objective function for the problem of Figure 1.

$$
\begin{aligned}
&Volume = \\
&\quad \left(1.14\ 10^{13}x - 5.66\ 10^{9}x^2 + 8.16\ 10^{5}x^3 + \right. \\
&\quad 3.28\ 10^{13}y - 3.26\ 10^{9}xy + 2.44\ 10^{5}x^2y - \\
&\quad \left. 6.70\ 10^{9}y^2 + 8.16\ 10^{5}xy^2 + 2.44\ 10^{5}y^3 - 4.08\ 10^{16}\right) / \\
&\quad \left(1.28\ 10^{1}xy - 2.56\ 10^{4}x + 2.56\ 10^{4}y - 6.40\ y^2 - 2.56\ 10^{7}\right)
\end{aligned}
$$

Table 6: Quadratic approximations of the gradient

$$
\begin{aligned}
&Quadratic\ Approximation\ of\ the\ gradient = \\
&\quad -1060187.5 + 159.4\ x + 0.05x^2 - 68.7\ y + 0.2\ x\ y - \\
&\qquad 0.00005\ x^2\ y + 0.03\ y^2 - 0.00003\ x\ y^2 + 5.9\ 10^{-9}\ x^2\ y^2, \\
&\quad -2909656.25 + 1860.4375\ x - 0.3\ x^2 + 1207.6\ y - 0.8\ x\ y + \\
&\qquad 0.0001\ x^2\ y - 0.14\ y^2 + 0.00009\ x\ y^2 - 1.5\ 10^{-8}\ x^2\ y^2
\end{aligned}
$$

obtained at compile time of the gradient of the volume function in Table 5 is shown in Table 6.

4 Experiments

To test the efficacy of this approach, we have solved a series of design problems using an implementation based on Mathematica [Wol88], and we have measured the impact of the compilation stages on the evaluation of the objective function, on the optimization task, and on the reliability of the optimization method. The measurements presented are averages over five randomly generated designs and, for each design, over 25 randomly generated starting points.

Objective function. The objective function of each design problem was evaluated in four different ways and, for each of them, we averaged the CPU[4] time over the different designs and starting points. The volume was first computed using the traditional, naive, numerical procedure with the method of joints. We then compiled the designs incorporating, in three successive stages, topological information, the givens of the problems, and the stress

[4]The examples were run on a NeXT Cube with a 68030 board.

state. Figure 5 shows the time (per 100 runs) to evaluate the objective function at the various compilation stages. The biggest speedup was obtained with the numerical substitution of values into the symbolic closed form expression obtained and with the specialization to stress states. This suggests that the gain is related to the elimination of arithmetic operations from the original numerical problem.

Optimization. As indicated in Section 1, the running time of the optimizers is influenced by the number of function calls and by the time for each function evaluation. To present the benefits of our approach on the optimization task, we have experimented with two optimization algorithms (a) an optimizer based on Powell's method that does not require gradient information and (b) the version of conjugate gradient descent [P+88] provided by Mathematica. The graphs in Figures 6 and 7 report, respectively, the number of objective-function calls and the overall CPU time for each optimizer. The values connected by solid lines correspond to cases where the optimizer had no gradient information, while the values connected by dashed lines indicate averages utilizing the conjugate gradient descent method with alternative approximations for the gradient vector.

As expected, the number of evaluations remains constant throughout the compilation stages when the non-gradient is used, while it decreases drastically when we switch to the gradient-based optimization method. The overall CPU time (Figure 7) steadily decreases as well. For the non-gradient method, the decrease is due to the progressive simplification of the objective function itself, so that it is cheaper to evaluate. When we switch to the gradient method, there is initially no speedup at all, because the cost of evaluating the full gradient offsets the decrease in the number of times the objective function must be evaluated. However, additional speedups are obtained by approximating the objective function as a quadratic and as a linear function (by truncating its Taylor series).

We have found experimentally that there is no appreciable difference between the minima reached using the full gradient vector and the minima computed using quadratic approximations of the partial derivatives. However, the precision of the results obtained with the linear approximation is significantly reduced. Depending on the application, this trade of accuracy for speed may be acceptable. If not, the quadratic approximation should be employed.

Another possibility is to employ the linear approximation for the first half

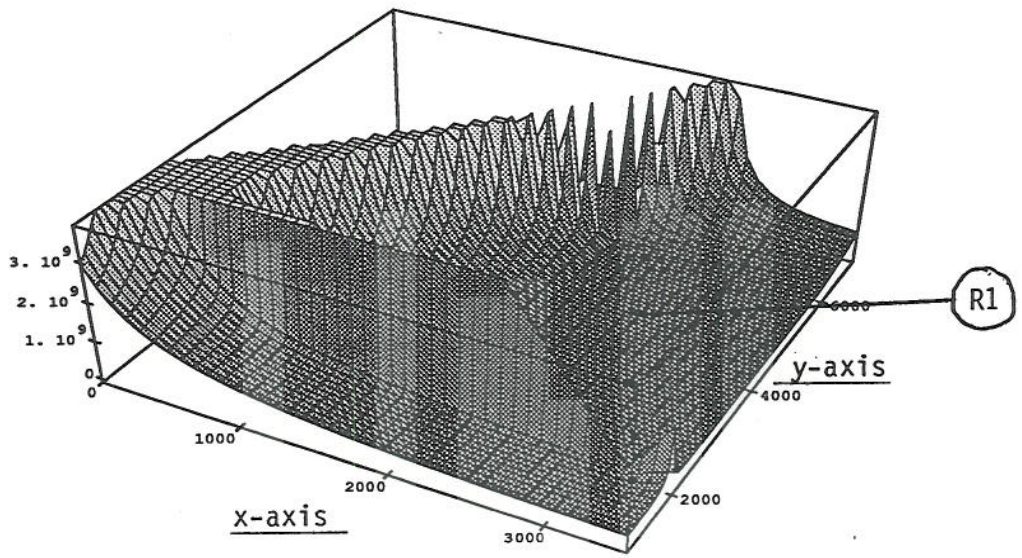

Figure 4: Volume of the structure

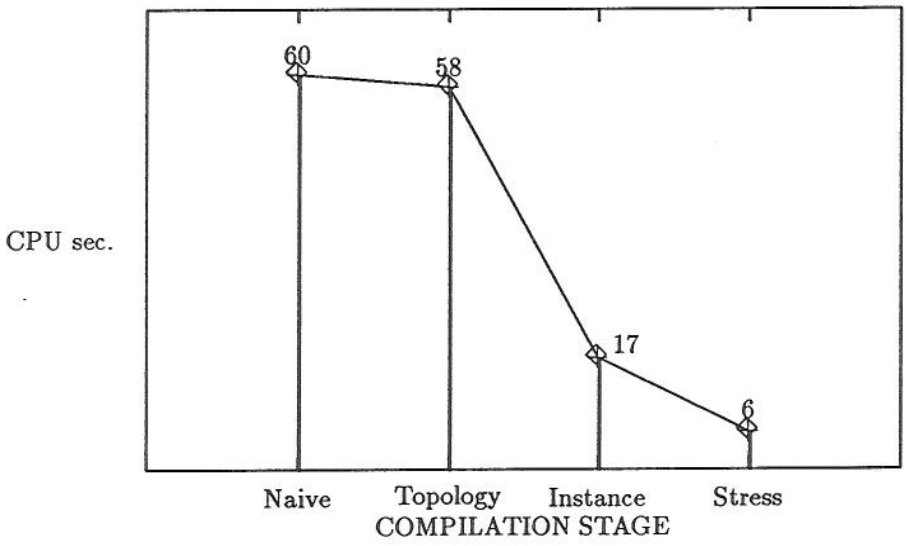

Figure 5: Average CPU time per function evaluation.

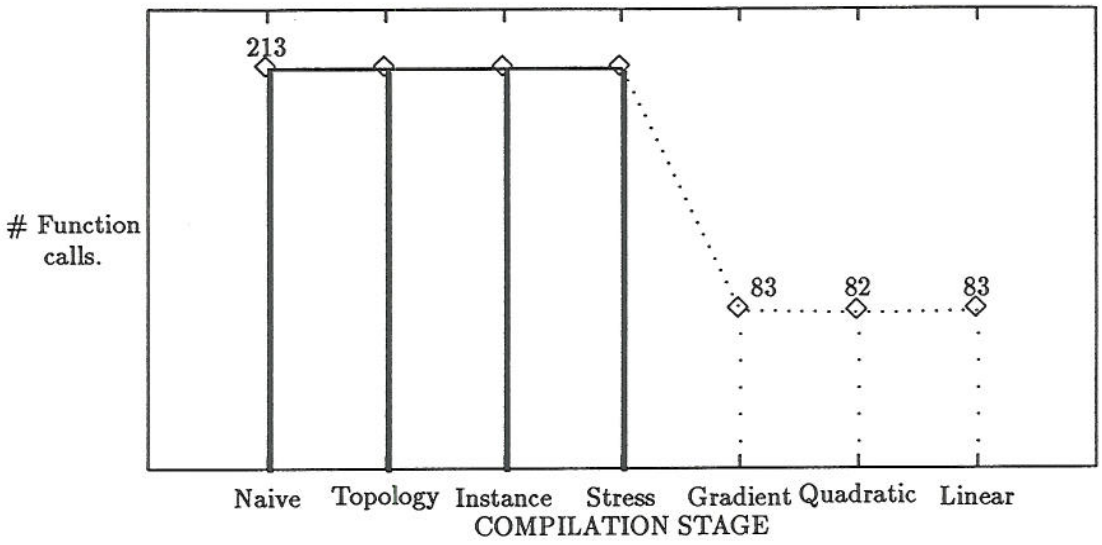

Figure 6: Average number of function calls.

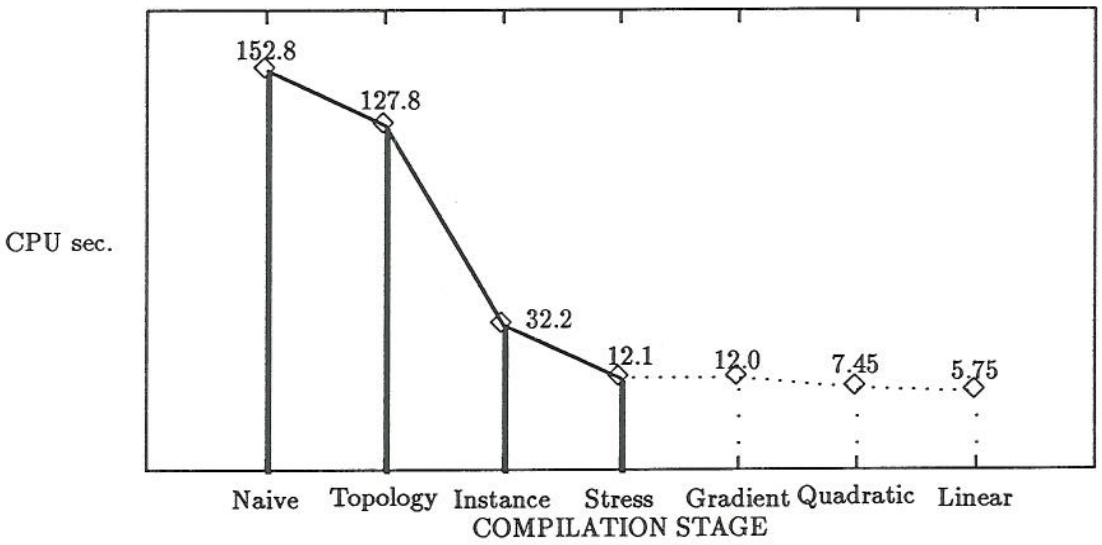

Figure 7: Average CPU time.

of the optimization search, and then switch to the quadratic approximation once the minimum is approached. In other words, the linear approximation can be applied to find a good starting point for performing a more exact search.

Reliability. An optimization method is reliable if it always finds the global minimum regardless of the starting point of the search. Unfortunately, as shown in Figure 4, the objective function in this task is not unimodal, which means that simple gradient-descent methods will be unreliable unless they are started in the right "basin." It is the user's responsibility to provide such a starting point, and this makes numerical optimization methods difficult to use in practice.

From inspecting graphs like Figure 4, it appears that, over each region corresponding to a single stress state, the objective function is unimodal. We conjecture that this is true for most of 2-D structural design problems. This means that optimization can be started from *any* point within a stress state, and it will always find the same minimum. If this is true, then our "divide-and-conquer" approach of searching each stress state in parallel will be guaranteed to produce the global optimum.

We have tested these hypothesis by performing 20 trials of the following procedure. First, a random starting location was chosen from one of the basins of the objective function that did not contain the global minimum. Next, two optimization methods were applied: the non-gradient method and the conjugate gradient method. Finally, our divide-and-conquer method was applied using, for each of the specialized objective functions V_j, a random starting location that exhibited the corresponding stress state. In all cases, our method found the global minimum while the other two methods converged to some other, local minimum.

5 Concluding Remarks

Our overall strategy for speeding up numerical optimization methods relies on successive specializations and simplifications of the objective function. This paper has described two specialization steps: (a) specializing the objective function by incorporating the invariant aspects of the particular problem and (b) splitting the objective function into special cases based on stress states.

In a companion paper [CD89], we illustrated another source of constraining knowledge that can be incorporated into the objective function. In that

paper, inductive learning methods were applied to discover regularities in the solutions found by numerical optimization. These regularities apply to particular classes of problems and include constraints such as $\alpha_1 = \alpha_2$ (i.e., two angles must be equal) and constraints that a member must be tangent to a "forbidden region" (a region into which the structure must not intrude).

When regularities of this kind are discovered, they too can be incorporated into the objective function. This often has the effect of reducing the number of independent variables, and hence, the dimensionality of the search space.

The benefits of all of these forms of specialization are great. First, the cost of evaluating the objective function is reduced. Second, the specializations make it possible to obtain differentiable closed forms for the objective function. This allows us to apply gradient-directed optimization methods, which generally require fewer evaluations of the objective function to find the optimum. Third, the specializations create opportunities for parallel execution of the optimization calculations. Existing numerical optimization procedures are inherently serial and contain almost no parallelism.

We are currently attacking the following open problems. First, not all of the 2^m load paths make physical sense. We have developed rules that can exploit this to prune useless load paths. Second, we hope to prove our unimodality hypothesis. This would provide a proof of correctness for our divide-and-conquer schema. Finally, we want to tackle the problem of selecting a good topology. Topological optimization is nearly impossible to perform using current numerical methods. However, if we can find reasonable approximations for the special case objective functions, we believe it will be possible to bound the weight of an optimal design for a given topology without performing the complete optimization process. This would allow us to determine the conditions under which one topology will be lighter than another. Such topology optimization rules would provide a valuable tool for mechanical designers.

Acknowledgments

The authors wish to thank Dr. David G. Ullman and Dr. Prasad Tadepalli for comments on drafts of the paper, Dr. Igor Rivin for information on the internals of Mathematica, and Dr. Jerry Keiper for insights into FindMinimum[] in Mathematica version 1.2.

References

[Bra88] Walter Braudaway. Constraint incorporation using constrained reformulation. Tech.Rep. LCSR-TR-100 Computer Science Dept., Rutgers Univerisity, April 1988.

[CD89] Giuseppe Cerbone and Thomas G. Dietterich. Inductive and numerical methods in knowledge compilation. In *Proceedings of the Workshop on Change of Representation and Problem Reformulation*, 1989.

[Ell89] Thomas Ellman. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163–222, 1989.

[Fri71] L.R. Friedland. *Geometric Structural Behavior.* PhD thesis, Columbia University at New York, N.Y., 1971.

[Gor78] James E. Gordon. *Structures: or, Why things don't fall down.* Plenum Press, New York, 1978.

[Min88] Steve Minton. Empirical results concerning the utility of explanation-based learning. In *Proceedings AAAI*, 1988.

[P+88] William H. Press et al. *Numerical Recipes in C: the art of scientific computing.* Cambridge Univerisity Press, Cambridge, 1988.

[PS70] A.C. Palmer and D.J. Sheppard. Optimizing the shape of pin-jointed structures. In *Proc. of the Institution of Civil Engineers*, pages 363–376, 1970.

[PW88] Panos Y. Papalambros and Douglass J. Wilde. *Principles of optimal design: modeling and computation.* Cambridge University Press, 1988.

[Van84] Garret N. Vanderplaats. *Numerical Optimization Techniques for engineering design with applications.* New York: McGraw Hill, 1984.

[Wol88] Steven Wolfram. *Mathematica.* Wolfram Research, 1988.

[WS84] Chu-Kia Wang and Charles G. Salmon. *Introductory Structural Analysis.* Prentice Hall, New Jersey, 1984.