GENETIC ALGORITHM OPTIMIZATION METHODS IN GEOMETRICAL OPTICS

by

NEAL C. EVANS

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

BIRMINGHAM, ALABAMA

1999

ABSTRACT OF DISSERTATION
GRADUATE SCHOOL, UNIVERSITY OF ALABAMA AT BIRMINGHAM

Degree Ph.D Program Physics

Name of Candidate Neal C. Evans

Committee Chair David L. Shealy

Title Genetic Algorithm Optimization Methods in Geometrical Optics

This dissertation explores the use of machine learning techniques, concentrating specifically on genetic algorithms (GAs), for solving beam shaping problems. In order to judge the effectiveness of this optimization-based method, four increasingly difficult beam shaping problems are solved. All four of these problems involve using a Gaussian input beam to a uniformly illuminate either spherical or planar surfaces some distance away. A computational method, which builds upon proven ray-tracing techniques, is developed for determining irradiance profiles. This method is the key to quantifying the efficacy of a beam shaper in terms of a merit function. When this merit function is coupled with a GA, an optimization technique can be employed.

The GA is able to find a satisfactory solution for all four cases in a significant but reasonable amount of time. This is particularly interesting since the GA requires little (often no) user input once the problem is started. In fact, in the last example, the GA is presented with a very general problem, and is allowed to determine the actual form of the system required to solve the problem, much as a human designer would. These examples demonstrate that the GA optimization-based method works, although the first two problems presented here can be solved in more general ways using analytical methods. With a general analytical solution, particular cases can be solved rapidly. However, the third and fourth examples illustrate two problems of such complexity that

analytical methods become difficult, if not impossible, to apply. The most promising applications of GAs lie in these areas.

DEDICATION

The light wraps around you in its mortal flame.

Abstracted pale mourner, standing that way

Against the old propellers of the twilight

That revolves around you.[1]

For Finley, for her undying love and support.

And, for my family, most of all my parents, for providing me with so much.

ACKNOWLEDGEMENTS

I am grateful to my advisor, Dr. David L. Shealy, for the many hours of personal conversation, guidance and teaching. His example has inspired me to become a better student of physics, as well as a better person. I appreciate his encouraging me to pursue those things which excited me, while insuring that I harness these indulgences to produce something of value. I am also most thankful to Dr. Joseph G. Harrison, Dr. Yogesh Vohra, Dr. Chris Lawson, Dr. James Buckley and Dr. Ian Knowles for serving on my committee and providing me with many fruitful avenues of exploration. I also wish to thank Dr. Vladimir Oliker of the Emory University Department of Mathematics and Computer Science, who provided me with my first laser-shaping problem: a two-mirror reflector. I also am indebted to Ken Baker of Optimetrix, 13659 Victory Blvd., Van Nuys, CA, 91401, with whom I worked to produce the first problem presented in this work, the laser shaper/projector system.

I am thankful for the funding provided to me by the U.S. Department of Education's GAANN program, which supported my research and tenure at UAB. Also, Optical Research Associates, 3280 E. Foothill Blvd., Pasadena, CA, 91107, has generously provided UAB the use of CODE V, an optical design and testing package, for research-related endeavors at a substantially discounted educational price. Without CODE V, this work would not have been possible.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

Page

LIST OF FIGURES (Continued)

Page

INTRODUCTION

Recently, the application of machine learning techniques including Neural Networks and Genetic Algorithms (GAs) to optimization problems has blossomed. Such techniques hold great promise not only because of their extraordinary efficiency and flexibility but also because they potentially allow the solution of previously intractable problems. So long as a fitness landscape[2] can be well-defined, a GA can be unleashed to roam this territory in a incessant search for the best solution. The GA must not, however, be characterized as a mindless automaton that wanders aimlessly about this terrain. Indeed, the essence of its value lies in the fact that the "...genetic algorithm [can] yield computer-based complex adaptive systems that can evolve strategies that no human being ever devised."[3]

Though there are numerous variations of GAs, they all share a central theme: their search strategy borrows concepts from natural selection and genetics.[4] Once presented with a specific optimization problem, the GA produces a set of potential solutions. These solutions are referred to as 'organisms' and a set of organisms is a 'generation.' GAs typically start with a randomly distributed seed generation, $G(0)$. For each generation $G(t)$, a new generation, $G(t+1)$ is produced based on the strengths and weaknesses of $G(t)$. Organisms are represented by a single string, or chromosome, which is built from the values of the parameters to be optimized for a particular problem.[5] These techniques endow GAs with several unique features, as described by Goldberg:[6]

1. GAs work with a coding of the parameter set, not with the parameters themselves.
2. GAs search from a population of points, not a single point.
3. GAs use payoff …[merit function] information only, not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transitions rules, not deterministic rules.

This evokes the intriguing thought of employing GAs to find solutions to problems in optics and optical design where analytical methods are difficult to apply and other optimization techniques are extremely inefficient or fail to yield good solutions altogether. As a first step, one must develop a GA optimization method and apply it to several well-understood problems. The key to this "proof-of-principle" stage lies in the fact that these problems have been attacked from a number of different perspectives. Not only does this provide a basis for judging the efficiency of the GA relative to other optimization techniques, but also this answers to the fundamental question "Does this method work?" For the applications presented herein, a GA method is developed that can be used to design laser beam-shaping systems that convert Gaussian input irradiance profiles to other specified profiles, typically uniform output profiles.

The laser beam-shaping system problem can be solved by a number of different methods, some of which are numerical and some of which are analytical. Some even employ a combination of both. Also, there are several different classes of beam-shaping systems, the most popular being those using diffractive elements[7,8,9,10] and those using refractive elements.[11,12] Reflective systems have also been produced.[13] The GA method can be used to optimize most systems of the above classes, which will be demonstrated in this word by the solution of several laser profile-shaping problems. The first three problems define the 'proof-of-principle' stage. Once it has been demonstrated that indeed the GA method can find solutions to these problem in a reasonably efficient

manner, the GA will be used to produce two entirely unique solutions to a difficult problem, which is presented in the last sections.

Scope of Applications

For the first problem, the GA determines the shape of one surface of a beam-shaping element such that the wavefront of a beam entering the system is modified to have a uniform irradiance profile on a surface some distance away.[14] To increase the complexity of the problem a bit, the beam is shaped such that it is uniform on a *spherical* surface. Thus, the system is diverging and the non-paraxial aspects of the system must be accounted for in subsequent irradiance calculations. A similar problem has recently been addressed in the literature using diffractive elements and a parametric optimization method.[15]

For the second problem, the GA is given two aspherical surfaces to shape, each respective surface being part of a separate shaping element. The GA must do this with the constraints that the outgoing beam is parallel to the optical ($Z-$) axis and that it has a specified radius. This problem is designed to mimic the system designed by Jiang, Shealy and Martin[16]. This should make for an interesting comparison of the efficiency of the two methods, in addition to demonstrating whether multiple solutions to the problem exist.

The final two problems inspired by a gradient-index shaping system designed by Wang and Shealy.[17] For these problems, the GA not only must determine shaping attributes such as element thickness and surface shape, but also must choose gradient glass types from a catalog. Since the glass type can only be chosen from a finite set of values, the parameter that describe the glass type is discrete. Many conventional optimization techniques work in a continuous parameter space, since they are often

driven by first- and second-order derivatives. The ability to choose from discrete parameters is a particularly powerful feature of the GA, relative to other optimization codes. In the final application, the GA is given creative latitude in determining that actual makeup of solution to the given problem. GA method used here is not a unique application in optics. Indeed, it should be noted that several other examples of GA-designed systems can be found in the optics literature.[18,19] In the next sections, the fundamental principles governing GA optimization are introduced by describing there application to the problems described above.

Computational Methods for Irradiance Calculations via Ray-trace Methods

Fundamental to laser beam-shaping computations is a fast, accurate means of determining irradiance (energy per unit area per unit time) profiles at different locations in an optical system. To do this, one must start with first principles: energy must be conserved in a non-dissipative optical system. This principle is mathematically expressed in the form of the energy conservation law. The energy conservation law has broad application, from designing reflective beam shapers via analytical differential equation methods to the development of finite-element mesh methods for the design of beam-shaping holograms.[20,21,22] In order to employ the energy conservation law, the concepts of rays and wavefronts must be developed as well as a description of how rays and wavefronts traverse an optical system. These concepts are fundamental to geometrical optics, and many discussions on this subject can be found in the literature.[23,24,25,26] In fact, the derivation presented below follows closely a discussion developed by Shealy.[27] Remembering that the ultimate goal is to describe the irradiance on defined surface, the optical field must be determined throughout the system. As described by Shealy, the optical field is a local plane wave solution of

Maxwell's equations for an isotropic, non-conducting, charge-free medium. It is a solution to the scalar wave equation[28, 29]

$$\left(\nabla^2 + n^2 k_0^2\right) e(\mathbf{r}) = 0\,, \tag{1}$$

where $e(\mathbf{r})$ represents the components of the electric field at any point $\mathbf{r}$, n is the index of refraction at $\mathbf{r}$, and k_0 is the wave number in free space. The wave number is described as follows:

$$k_0 = \omega/c = 2\pi/\lambda_0\,, \tag{2}$$

where ω is the frequency of the wave, c is the speed of light, and λ_0 is the wavelength of incident light. Assume that a solution to Eq. (1) can be written as

$$e(\mathbf{r}) = e_0(\mathbf{r}) \exp\left[ik_0 S(\mathbf{r})\right] \tag{3}$$

where $e_0(\mathbf{r})$ and $S(\mathbf{r})$ are unknown functions of $\mathbf{r}$. Substituting Eq. (3) into Eq. (1) and performing the indicated operations, one finds that the following conditions must be satisfied by $S(\mathbf{r})$ and $e_0(\mathbf{r})$ (neglecting term proportional to $1/k_0^2$):

$$\left(\nabla S\right)^2 = n^2 \text{ and} \tag{4}$$

$$2e_0 \nabla S \cdot \nabla e_0 + e_0^2 \nabla^2 S = 0\,. \tag{5}$$

Equation (4) , known as the eikonal equation, is a fundamental relation in geometrical optics. Surfaces described by

$$S(x, y, z) = const. \tag{6}$$

are known as the geometrical wavefronts. The surfaces represent constant phase solutions to Eq. (3). The concept of optical rays is derived from geometrical wavefronts:

rays always are normal the wavefronts in isotropic media. A unit vector normal to the wavefront and along a ray at the point $\mathbf{r}$ is given by

$$\mathbf{a}(\mathbf{r}) = \frac{\nabla S(\mathbf{r})}{\left|\nabla S(\mathbf{r})\right|}. \tag{7}$$

Coupling this with the eikonal equation, one finds

$$\mathbf{a}(\mathbf{r}) = \frac{\nabla S(\mathbf{r})}{n(\mathbf{r})}. \tag{8}$$

Equation (8) defines a ray vector at the point $\mathbf{r}$.

As illustrated in Ref. 26, Eq. (5) can be recognized as one form of the geometrical optics intensity law for the propagation of a bundle of rays. To see this, one starts by rewriting Eq. (5) using the vector identity $\nabla\cdot(f\,\mathbf{v}) = f\,\nabla\cdot\mathbf{v} + \mathbf{v}\cdot\nabla f$, which results in

$$\nabla\cdot\left(e_0^2\,\nabla S\right) = \nabla\cdot\left(e_0^2 n\mathbf{a}\right) = 0. \tag{9}$$

Recognizing that the energy density of a field is proportional to the square of the field amplitude e_o^2 and that the intensity I is equal to energy density of the field times the speed of propagation within medium, then Eq. (9) can be written as

$$\nabla\cdot\left(I\,\mathbf{a}\right) = 0. \tag{10}$$

Multiplying Eq. (9) by the constant $\left(c/4\pi\right)$ for CGS units gives the correct dimensions for intensity.[30] Equation (10) expresses conservation of radiant energy for non-conducting medium. Integrating Eq. (10) over a tube surrounding a bundle of rays[31] as illustrated in Fig. 1 gives and applying Gauss' theorem on that result yields

$$I_1\,dA_1 = I_2\,dA_2. \tag{12}$$

Equation (12) expresses conservation of energy along a ray bundle between any two surfaces intersecting the beam and is a basic equation used to the laser beam shaping systems presented herein. To employ energy conservation for the systems here, the irradiance (irradiance is used interchangeably with intensity) profile of a bundle of rays striking the input pupil is defined by a radially symmetric function $\sigma(\rho)$. These rays propagate through the beam profile-shaping system (the 'black box') and exit to strike the Output Surface. The irradiance distribution on the Output Surface is represented by the function $u(\mathrm{P})$. Assuming no energy is dissipated by the system, Eq. (12) (re-written here as it is before applying Gauss' theorem) must be satisfied:

$$E = \int_I \sigma(\rho)\left(\hat{\mathbf{n}}^{in}(\rho)\cdot\mathbf{v}^{in}(\rho)\right)da = \int_O u(\mathrm{P})\left(\hat{\mathbf{n}}^{out}(\mathrm{P})\cdot\mathbf{v}^{out}(\mathrm{P})\right)dA, \tag{13}$$

where E is the total energy entering the system and I and O are the Input Plane and Output Surface, over which the respective integrations occur. The irradiance function on the Input Plane, $\sigma(\rho)$ is the same as I_1 in Eq. (12) and irradiance function on the Ouput Surface, $u(\mathrm{P})$ is I_2. Also, $\hat{\mathbf{n}}^{in}(\rho)$ and $\hat{\mathbf{n}}^{out}(\mathrm{P})$ are the normal vectors on the input and output surfaces, respectively. $\mathbf{v}^{in}(\rho)$ and $\mathbf{v}^{out}(\mathrm{P})$ are unit vectors along the direction of an individual ray (striking the input surface at radial height ρ, and the output surface at radial height P) at the input and output surfaces, respectively. See Fig. 2 for further elaboration of terms in Eq. (13). da and dA are derived below.

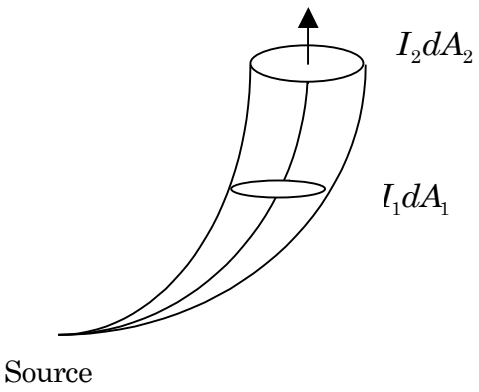

Fig. 1. Illustration of Conservation of Energy with a Bundle of Rays (from Ref. 27).

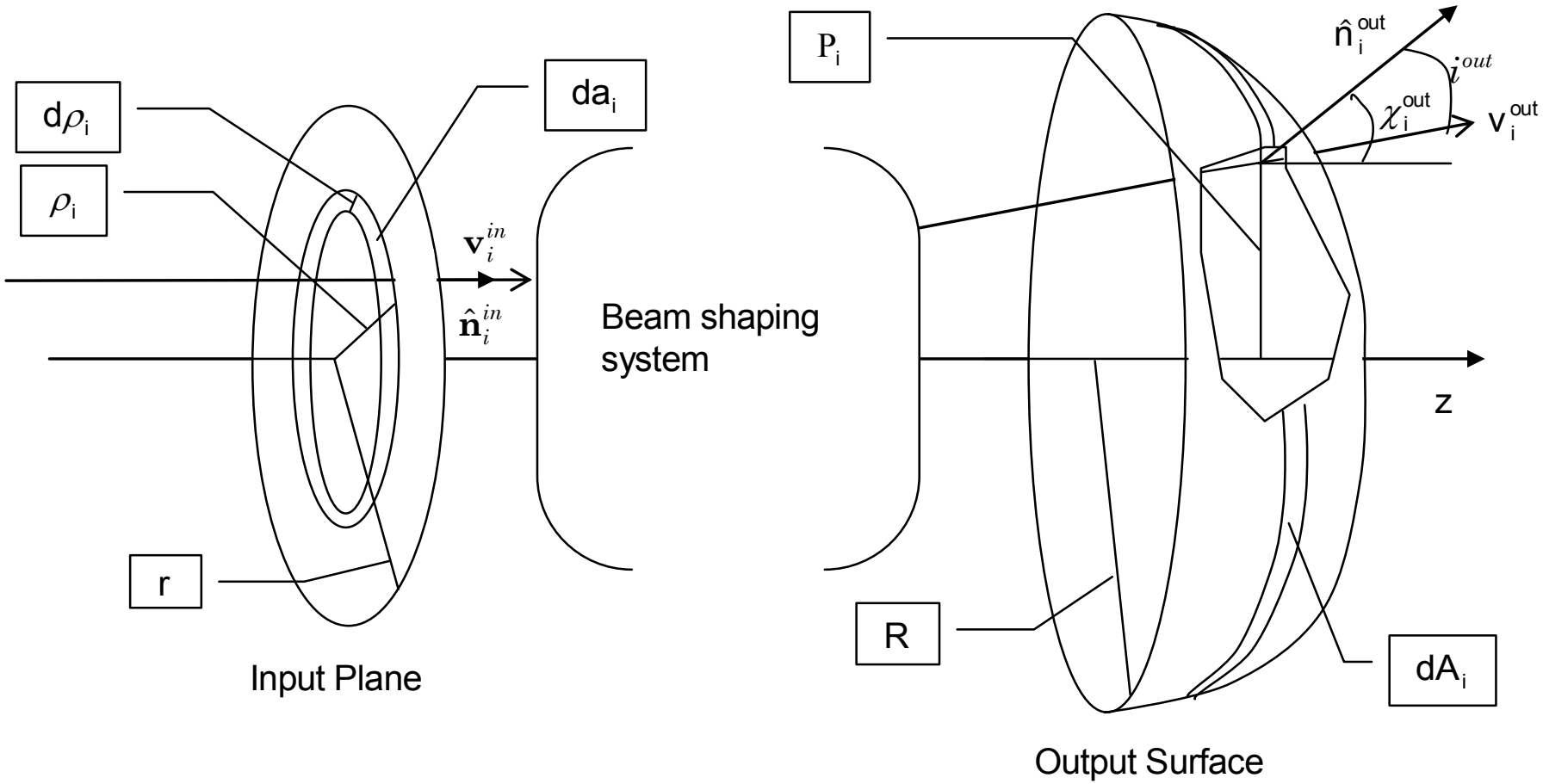

Fig. 2. Beam expander with Input Plane and Output Surface. The beam profile is shaped to be uniform on the Output Surface (from Ref. 14).

Balancing the radiant energy striking differential ring da with the radiant energy exiting differential ring dA, as required by Eq. (13), one can see that

$$u(\mathrm{P}) = \sigma(\rho)\,\frac{\left(\hat{\mathbf{n}}^{in}(\mathrm{P})\cdot\mathbf{v}^{in}(\mathrm{P})\right)da}{\left(\hat{\mathbf{n}}^{out}(\mathrm{P})\cdot\mathbf{v}^{out}(\mathrm{P})\right)dA}. \tag{14}$$

To determine the ratio da/dA in Eq. (14), N rays are traced through the system, where N is a reasonably large number, though not so large as to be computationally expensive. For this application, $N = 200$ is chosen, which gives adequate resolution for the input and output profiles. Each ray enters parallel to the optical (Z-) axis at a specified height, ρ_i, where the set of ρ_i are distributed equally across the radius of the Input Plane according to the following function:

$$\rho_i = \left(\frac{r}{N}\right) i, \quad i = 0 \ldots N \tag{15}$$

Each ray will exit the system and strike a point on the Output Surface, as shown in Fig. 2. At the point where the ray strikes the Output Surface, P_i, the axial distance from the optical axis, and χ_i, the angle between the unit vector normal to the Output Surface at the intercept point and the optical (Z-) axis, are measured. Thus, an array with $3N$ members (3 columns: ρ_i, P_i, χ_i, and N rows) is populated from ray trace data.

One can see in Fig. 2 that da_i is given by $2\pi\,\rho_i\,d\rho_i$, where

$$d\rho_i = \rho_i - \rho_{i-1}. \tag{16}$$

The definition of $d\rho_i$ in this manner is arbitrary; definitions such as $d\rho_i = \rho_{i+1} - \rho_i$ or $d\rho_i = \rho_{i+1} - \rho_{i-1}$ would be just as effective. Furthermore, the subscript i is introduced

to emphasize the numerical nature of the solution to the now discrete function in Eq. (14). Calculation of dA_i is somewhat more complicated, since the Output Surface is a not flat like the Input Plane. In general, dA_i is given by $2\pi \, \mathrm{P}_i \, dS_i$, where dS_i is found by referring to Fig. 3:

$$dS_i = \frac{d\mathrm{P}_i}{\cos \chi_i^{out}}. \tag{17}$$

Also, it is clear from Fig. 2 and Fig. 3 that

$$\hat{\mathbf{n}}^{in}(\rho) \cdot \mathbf{v}^{in}(\rho) = \cos(i^{in}(\rho)) \text{ and} \tag{18}$$

$$\hat{\mathbf{n}}^{out}(\mathrm{P}) \cdot \mathbf{v}^{out}(\mathrm{P}) = \cos(i^{out}(\mathrm{P})). \tag{19}$$

Combining these observations with Eqs. (14)-(17), one has

$$u(\mathrm{P}_i) = \sigma(\rho_i) \left(\frac{\cos(i_i^{in}) \, \rho_i (\rho_i - \rho_{i-1}) \cos(\chi_i^{out})}{\cos(i_i^{out}) \, \mathrm{P}_i (\mathrm{P}_i - \mathrm{P}_{i-1})} \right). \tag{20}$$

In the examples presented herein, the input irradiance is assumed to be Gaussian, measured in units of rays per unit area:

$$\sigma(\rho_i) = \exp\left(-\alpha \rho_i^{\,2}\right), \tag{21}$$

where α is a unitless quantity given by $-2\rho_i^2 / \rho_N^2$. Here, the beam waist of the incoming beam is expressed by ρ_N, and is defined as the radius of the circle where the irradiance drops to $1/e^2$ of the central irradiance. The N rays that are traced through the system are distributed uniformly over the Input Place according to Eq. (15). Though a variety of input profiles may be used with this method, a Gaussian input profile is chosen because it describes typical laser profiles when the laser is in the fundamental

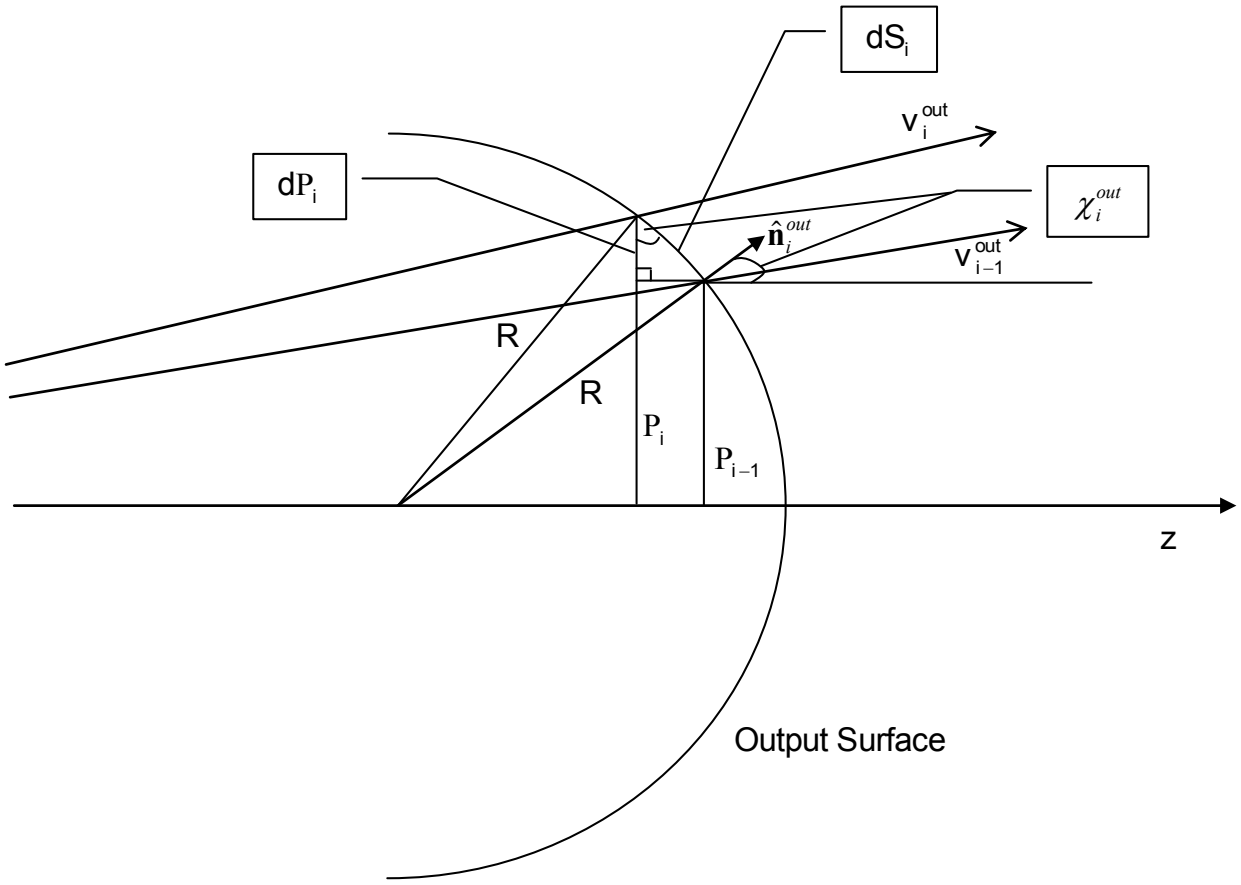

Fig. 3. Determination of dA_i. From the figure, one can see that $d\mathrm{P}_i = \mathrm{P}_i - \mathrm{P}_{i-1}$ and that $dS_i = d\mathrm{P}_i / \cos(\chi_i^{out})$ (from Ref. 14).

mode (TEM_{00}). Eq. (20) expresses the output beam irradiance in terms of the input beam irradiance times a ratio of areas expressing the beam expansion as a result of ray propagation through the optical system. Eqs. (20) and (21), along with the ray trace array, provide an accurate means of calculating the beam profile over any reasonable surface. The accuracy of this method has been verified by calculating the profiles for several benchmark systems.[16,17] Calculations of output beam profiles using Eq. (20) are in close agreement with the profiles given in the benchmark papers. Now, a merit function can be developed based on Eq. (20) which allows the GA to distinguish between systems with uniform intensity profiles (which are desired) and non-uniform profiles. This is done in the sections that follow.

THEORY OF OPTIMIZATION

Generally, the idea behind optimization is that one has some function f which may be evaluated easily, usually computationally. This function is expressed in terms of several variables which may be discrete or continuous in nature. One wishes to find the values of these variables which make f assume either its maximum or minimum value. The difficulty of the problem is related to whether one is searching for *local* extrema, of which there may be many, or the *global* extrema, which represent the absolute best solutions. The complexity of the problem is related to the number of variables which make up f, in addition to the ease with which f can be calculated.[32] The greater the complexity of the problem, the longer it takes to arrive at a solution. Thus, search algorithms which arrive at solutions quickly are to be coveted, which is evident by the voluminous amount of research regarding the subject present in the literature.[33,34] In this work, the GA search method is presented as one of these treasured methods, but it should be noted that other algorithms exist which produce similar, if not superior performance. Two popular alternative methods, simulated annealing and the Tabu search[35] are discussed below.

Overview of Iterative Computational Optimization Methods

Though there are myriad optimization techniques to choose from, methods such as GAs and Simulated Annealing are of particular interest because of their ability to solve combinatorial minimization problems. The key feature of such problems is that

one or more of the parameters that make up the merit function (which is to be maximized) are discrete, in the sense that they only can assume particular values from a pre-defined set of allowable values. Thus, instead of an $N-$dimensional space made up of N continuous parameters, one is presented with a parameter space whose complexity is factorially large—so large in fact that it cannot be completely explored.[36] Without a continuous merit function, concepts such as "downhill" and "uphill" lose their meaning and other optimization techniques, such as the Simplex Method,[37] can no longer be applied. For example, in the problem presented in Design and Analysis of a Gradient-Index Shaper and Design and Analysis of a Free-Form GA-Designed GRIN , the glass types of the lens elements in the system are chosen from a fixed set of gradient-index materials found in a manufacturer's catalog. It is here that GAs and Simulating Annealing techniques excel, though they also can be applied to problems that are purely continuous as well. In the literature, there are numerous examples of problems solved using these techniques,[38,39] as well as research that compares the performance of one or more of these methods on the same class of problems.[40,41,42,43] It seems that of the three methods discussed here, no one method is necessarily more efficient than the others, though it does appear that GAs and the Tabu search tend to arrive at solutions more quickly than Simulated Annealing methods, at least in the papers cited here.

It should be noted that several commercial optical design and analysis packages implement these techniques in their optimization routines to varying degrees. OSLO[44], for example, uses an 'adaptive simulating annealing' method. ZEMAX[45] and CODE V[46] also contain proprietary global optimization methods. The problem with these implementations, among other things, is that the merit functions in these packages are

oriented towards imaging systems (ZEMAX, however, allows for user-defined merit functions computed by macros or an external programming interface), limiting one's ability to manipulate the merit function for one's own purpose. Also, since the makers of these packages keep their optimization codes proprietary, one's ability to tweak those routines is all but eliminated. More ambitious goals like parallelization of the optimization code (see Parallelization of the Genetic Algorithm below) becomes extremely difficult, at best.

Genetic Algorithms

Since GAs are based on a biological paradigm, a lot of the GA nomenclature is borrowed directly from evolutionary biology. The reader may find it useful to have some of this jargon expressed in terms more familiar to the optics community. As discussed above, GAs produce a finite number of test solutions to a problem. Individually, these solutions are referred to as 'organisms' (or just 'individuals'), and collectively as a 'generation.' A 'generation' is essentially an iteration. With each iteration, the merit function , M , is evaluated for each member of a generation. There may be a few as five or as many as hundreds of individuals per generation, depending on the code used and how it is configured. For the applications explored here, there are typically five or ten individuals in a generation. A new generation of child systems is produced from the genetic material of the parent generation (the specifics of this process are described below). An individual's genetic code represents a particular system prescription. For example, in the Beam Shaper/Projector example presented in Design and Analysis of a Beam Shaper/Projector, the six parameters that collectively define one surface of the beam-shaping element are concatenated into a string (i.e., genetic code). Thus, with

each iteration, five or ten new system prescriptions are produced and their respective merit functions evaluated.

The GA method developed here is based on a Micro-GA code.[47,48,49] Micro-GAs have several features which distinguish them from other GA codes. The most prominent of these features is the fact that Micro-GAs can operate efficiently with small generation sizes (on the order of 10 individuals per generation). This is important for the applications presented here, since evaluation of the merit function for each individual is a very time-consuming process. The Micro-GA is able to work with small generation sizes by checking for 'stagnancy' in each generation it produces. Stagnancy is determined by taking the average value of the merit function for all the individuals in a generation, $\bar{M}(t)$, and comparing it to average merit functions values for N parent generations, $\bar{M}(t-1), \bar{M}(t-2), ..., \bar{M}(t-N)$. If these values do not differ significantly (a parameter which can be set in the Micro-GA, and is usually 'tweaked' at the beginning of a problem to produce the most efficient code), then the population is defined as stagnant. Essentially, when stagnancy is detected, the code assumes that the GA is stuck in a local minima and attempts to add some randomness to the process. When such a situation arises, the GA picks the best of the individuals in a generation, kills the remaining and replaces them with new, randomly-selected individuals in the child generation.

'Reproduction' is defined as producing a child generation of new individuals from the genetic material of a parent generation. The individual with the highest value of M in a particular generation is most likely to have its genetic material passed on to the next generation. The 'genetic material' for a particular individual is defined by concatenating the binary value for each parameter to be optimized into a binary string

(consists of only ones and zeros). See Fig. 4 for an example. New generations are created by a 'crossover operator', which swaps chunks of genetic material (strings) between two or more individuals in a generation. With the Micro-GA code, crossovers are done in a manner that maintains individual 'alleles'. An allele is the particular value that a parameter assumes, expressed in string format.[50] When a crossover occurs, the strings that represent alleles are not broken into pieces, but are transferred from one individual to another intact. Another operation the GA performs is 'mutation.' Here, the GA randomly selects one or several bits in an allele, and changes the state of these bits. Since the string is binary, this amounts to operating on the bit with 'Not' (not 0 = 1, not 1 = 0). Mutation add a built-in randomness to the GA method, which helps the GA avoid local minima. Because of the stagnancy-checking feature it employs, the Micro-GA allows one to avoid the constant tweaking of GA parameters (e.g., crossover and mutation rates) which is often necessary with conventional routines. The flowchart for this GA, referred to as the sequential GA, is shown in Fig. 5.

Parallelization of the Genetic Algorithm

Given that the total execution time for the problems discussed in this work is nearly seven hours, it is important to increase the efficiency of the GA method. This can be accomplished by having the code execute in parallel, something facilitated by the nature of the GA. Parallel implementations of GA codes are common in the literature, and several different strategies for parallelization exist.[51,52] The two most popular strategies are described below.

For the first strategy, it is interesting to note that, generally, the most computationally intensive step of the optimization process involves the evaluation of the merit function. In these applications, this involves calling optical simulation or optical

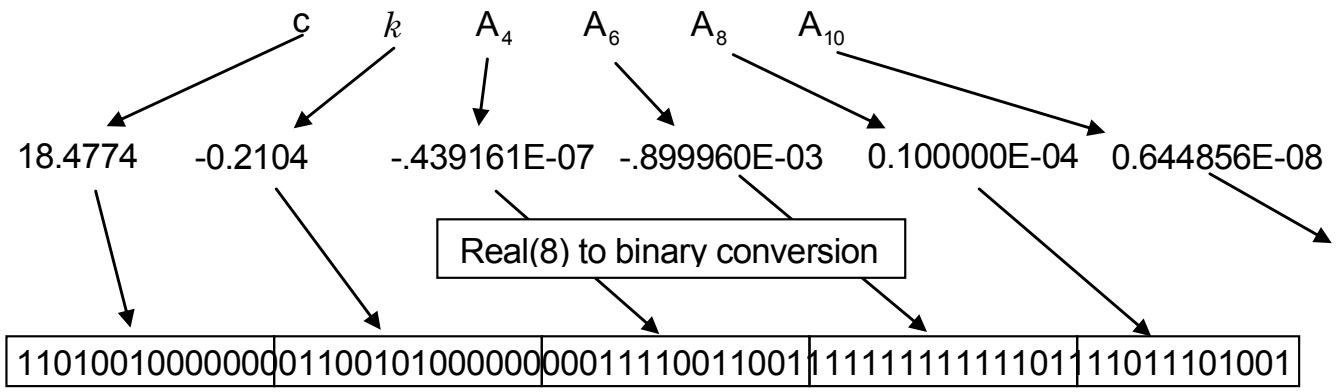

Fig. 4. Example of the genetic material for a single individual. Values (Real*8) for each parameter are converted into binary strings, which are in turn concatenated into one long string, the genetic material for that individual (from Ref. 14). Individual alleles (the values that $c, k, A_4, \ldots A_{10}$ assume, expressed in binary form) are kept intact when crossovers occur.

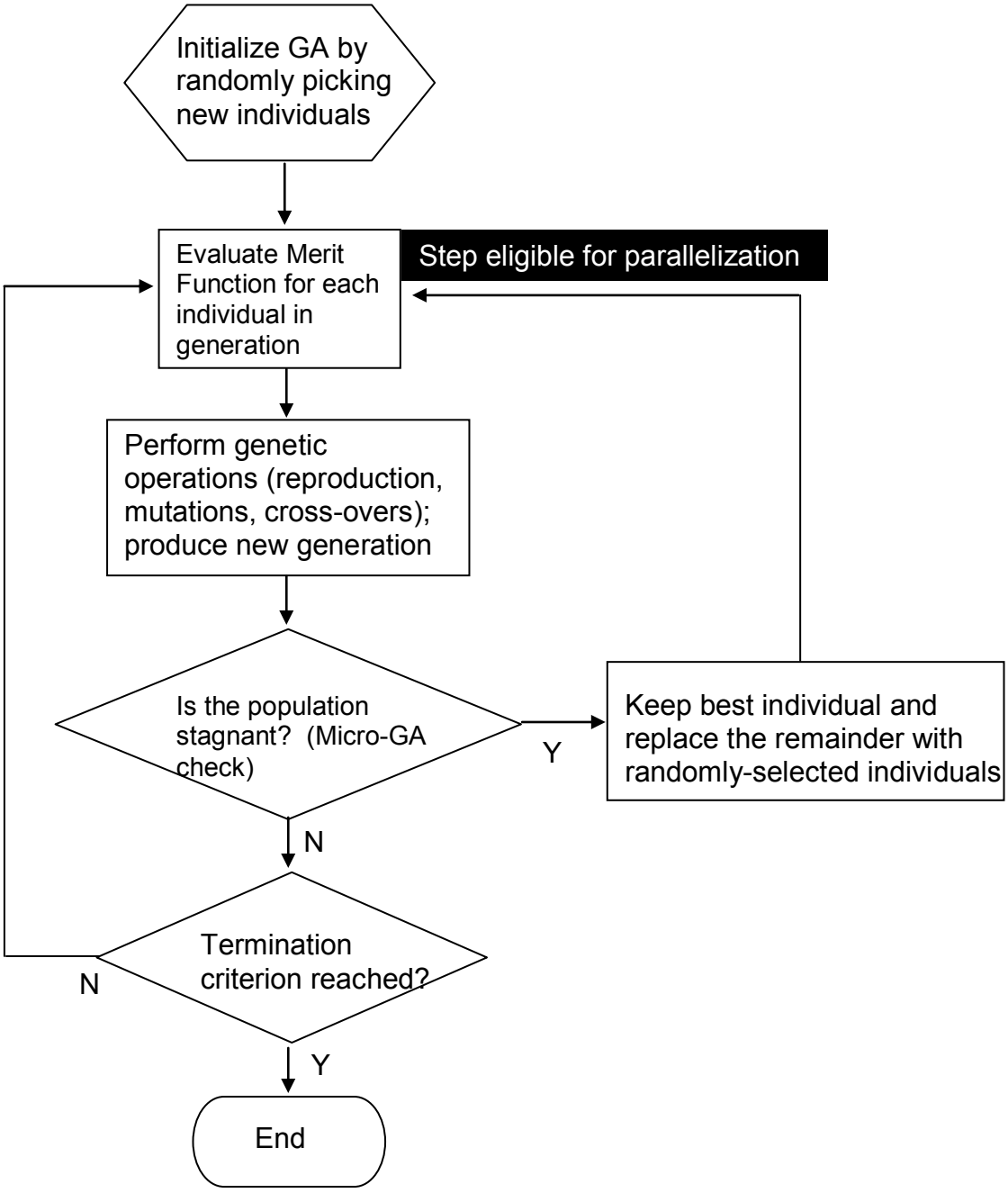

Fig. 5. Flowchart of the sequential micro-GA. For definitions of reproduction, mutations, cross-overs and stagnancy see Genetic Algorithms.

design packages, such as CODE V, which are external the Genetic Algorithm code itself. For example, in the problems presented herein, the evaluation of the merit function involves calling CODE V and tracing $N = 200$ rays for each system. The calculation time for a single generation with 10 individuals takes about 9 seconds (on a Sun Ultra 1 170 with 64M of RAM). Of this, about 8 seconds on average is spent in CODE V. The evaluation of the merit function step is an obvious candidate for parallelization. Thus, a potential parallel scheme is one where the GA executes sequentially on one machine, the master node, until it reaches the point where the merit function is to be evaluated. Here, the master initiates processes on each of the slave nodes. The slave nodes, in turn, evaluate the merit functions for individuals in the generation given to them in parallel and return the results to the master. Once all the merit functions are evaluated, the GA code runs normally on the master and executes all GA-related operations, producing the next generation. This parallel scheme, known as the Standard Parallel GA Paradigm, is shown in Fig. 6, along with an alternative parallel scheme, which is explained below.

An alternative scheme, the subpopulation parallel paradigm, essentially runs independent instantiations of the GA code on each node. With time, different nodes produce different best individuals with varying degress of fitness. The best individuals are periodically sent to a central bookkeeping node, the "integrator". The integrator finds the "best of the best" and distributes this champion to the other slave nodes, where the genetic material of this champion is assimilated by the local subpopulation. If there are enough nodes, each subpopulation can execute on a pod of machines using the Standard Parallel GA Paradigm outlined above. Thus, one must test the efficiency not

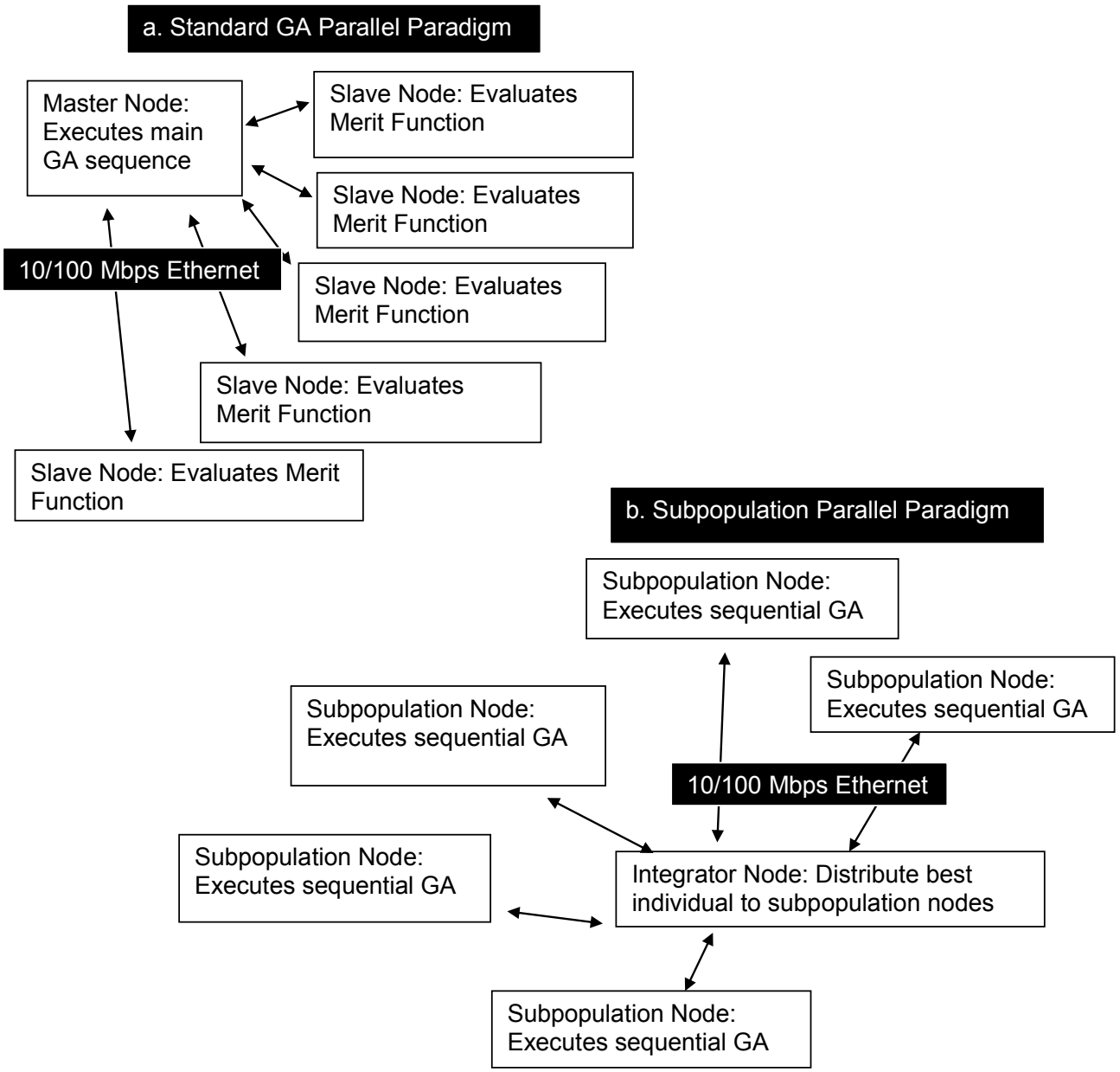

Fig. 6. Two parallel GA paradigms. In the first setup, the standard GA paradigm (a), the GA is executed sequentially on the master until the step where the merit function is evaluated. At this point, the merit function is evaluated in parallel on the slave nodes. In the second setup, the subpopulation parallel paradigm (b), the GA executes normally on several slave machines, which at pre-defined time send their best individuals to the integrator. The integrator then redistributes these individuals to the other slave nodes.

only of the two paradigms individually, but also a hybrid scheme that incorporates both paradigms.

One could run the parallel GA (PGA) on a inhomogeneous cluster of workstations on the 10/100 Mbps local-area network. Message passing among nodes could be accomplished using the MPI-2[53,54] libraries, making the PGA code easily-scalable to high-performance massively-parallel systems. For most research, however, scaling to parallel supercomputers is not feasible since evaluation of the merit function for common problems requires calling proprietary software packages such as CODE V, for which the source code is not (freely) available.[55] If problems are chosen where all source code is available, including that for evaluating the merit function, the GA can be ported to a supercomputer environment with (relatively) little modification.

APPLICATIONS

One key feature of the GA method is its broad applicability. Using the theory and tools developed above, one can adapt the GA to solve a multiplicity of problems. The problems presented below are chosen not only to demonstrate this advantage, but to do so while building a logical, concise method that builds from simple to more complex applications. Furthermore, these problems are chosen from current literature and provide a means to compare the solutions generated by the GA with solutions generated by other methods. Hopefully, this will provide insight into where the GA methods are appropriate and where they are of little advantage. The three problems chosen here are Design and Analysis of a Beam Shaper Projector, Design and Analysis of a Gradient-Index Shaper and Design and Analysis of a Two-lens Shaper.

Design and Analysis of a Beam Shaper/Projector

The general goal here is to modify the shape of a lens element (the 'shaping element') to uniformly illuminate a spherical surface some distance away. See Fig. 7 for a ray trace of the system. For this particular application, the use of a Genetic Algorithm is perhaps a tad overzealous, considering that other more established design methods could be employed to produce solutions both easily and efficiently. The goal, however, is a long-term one: the GA technique will be used to attack systems that are difficult to solve with more conventional methods. For example, certain holographic projection systems have fitness landscapes with 20 or more dimensions and extremely complex

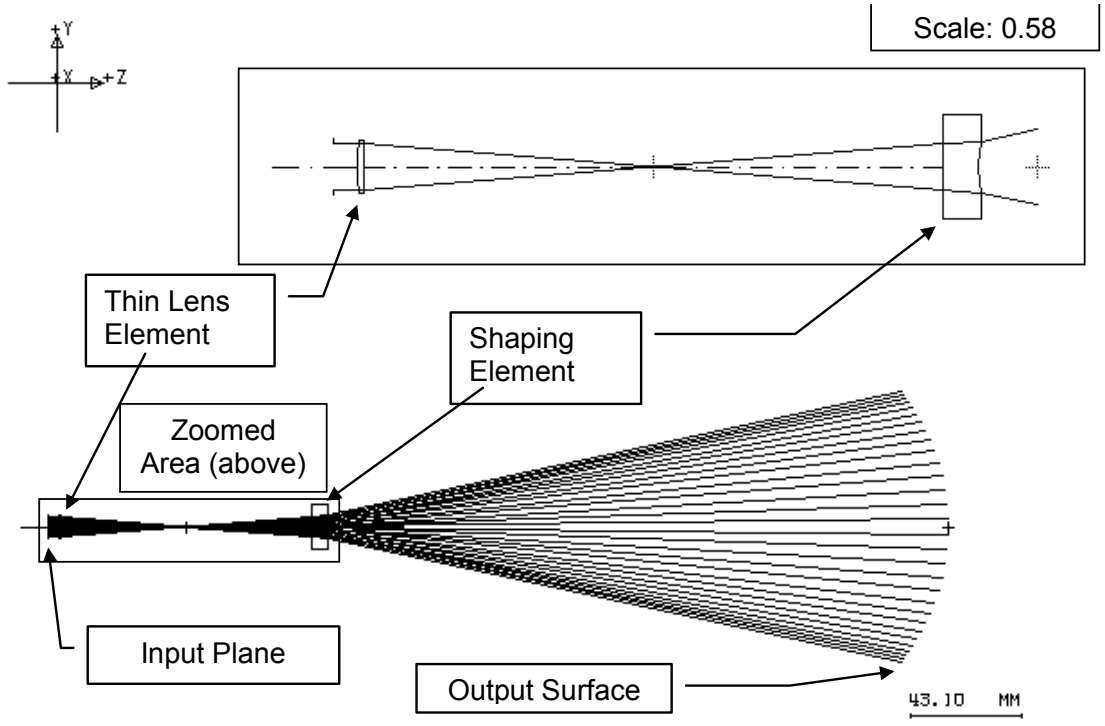

Fig. 7. Beam-shaping system with ray trace showing the density of rays increases at the periphery of the Output Surface, as one would expect to compensate for the Gaussian nature of the input beam. Both the thin lens element and the shaping element are shown. The shaping element is determined by the GA (from Ref. 14).

merit functions, making their solution with conventional methods very tedious.[56] For the short-term, it must be established that the GA technique produces good solutions in a reasonably efficient manner. This is accomplished by the application of the GA technique to simple, well-understood systems. The insights gained by this application will provide a picture of the fundamental mechanisms that govern this and the nuances involved in its proper implementation.

The application of a GA generally must satisfy two prerequisites. First, one must identify those parameters that fundamentally characterize the system. The parameters must be numerically quantifiable and the modification of these parameters should have direct consequence on the system itself. Second, one must identify those features of a system which best describe the fitness (or 'merit') of the system. This could be one particular attribute, such as focal length, or, on the other extreme, could involve the blending of many different attributes, each with different weights and measures of influence.

In the application of the GA to the beam profile-shaper, one surface of a refractive lens is modified such that the irradiance profile over a spherical image surface is uniform (that is, $u(\mathrm{P}) = \text{constant}$ in Eq. (14)). The rotationally-symmetric lens surface is characterized by the conventional surface equation used in optics:

$$z(h) = \frac{c\,h^2}{1+\sqrt{1-(1+k)\,c^2h^2}} + \sum_{j=2}^{5} A_{2j} h^{2j}, \tag{22}$$

where z is the sag of the surface, c is the curvature of the surface, k is the conic constant and $A_4 \ldots A_{\_\_}$ are aspherical deformation coefficients. z is an even function of h, the surface radial distance from the optical axis. The choice of this function allows the GA great flexibility in determining the shape of the lens surface, depending on the

number of deformation coefficients included in the optimization process. One might speculate that the lens surface must be highly aspherical, based on results from similar beam-shaping systems, such as that discussed in Ref. 16. Furthermore, it is desirable to provide the GA with a large (multi-dimensional) parameter space to explore, since this is where GAs are especially powerful. Thus, the GA is given six parameters to optimize: $c, k, A_4, A_6, A_8,$ and A_{10}.

Finally, the GA must be given a means of distinguishing between good systems and bad systems. In this application, a uniform irradiance profile over the Output Surface is desired. The Output Surface is a sphere with a radius of $R = 84.12\,\text{cm}$.[57] Furthermore, it is required that the exit pupil have a radius of 50 cm. To accomplish this, the following merit function is defined:[14]

$$M = \frac{1}{\mu}\exp\left[-s\left(50 - \mathrm{P}_N\right)^2\right], \tag{23}$$

where

$$\mu = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(u(\mathrm{P}_i) - \bar{u}\right)^2} \tag{24}$$

and

$$\bar{u} = \frac{1}{N}\sum_{i=1}^{N} u(\mathrm{P}_i). \tag{25}$$

$u(\mathrm{P}_i)$ is defined in Eq. (20). In Eq. (23), P_N is the radial height (as measured from the optical axis) of the marginal ray on the Output Surface, which defines the exit pupil in this case. The exponential function is chosen since, for this problem, one wants the merit function to peak sharply at $P_N = 50\,\text{cm}$. The exponential accomplishes this

nicely, but functions other than the exponential may have been chosen for the same purpose. s determines the sensitivity of the merit function to the exit pupil radius constraint: the smaller the value that s assumes, the more prominent the exponent becomes. In this example, s is set to 0.01. This value is adjusted on occasion while the GA is executing to insure that the pupil radius constraint is satisfied. In Eqs. (24) and (25), $\bar{u}$ is the "mean" of the values of the output intensity function, $u(\mathrm{P}_i)$ over N points on the Output Surface. As the beam profile on the Output Surface becomes more uniform, μ approaches zero, and M increases substantially. Also, as the exit pupil, which is measured by P_N, approaches $50\ \mathrm{cm}$, the value of M peaks as a result of the exponential in Eq. (23). This is illustrated in Fig. 8. Systems with the desired characteristics—a uniform beam profile on the Output Surface and an exit pupil of $50\ \mathrm{cm}$—will have higher values of M, which is precisely what is required. The GA will find those systems with the highest value of M.

The beam-shaping element consists of two surfaces. The first surface is flat and the second surface, of course, has its shape determined by the GA. The GA starts by randomly choosing, within pre-determined constraints, values for the six surface-shape parameters to be optimized. In each generation, ten individuals are produced (see section 0 for explanation of GA nomenclature). The parameters for each individual are passed to a ray-tracing routine (CODE V is used for ray tracing in this application) where N rays are traced and the value of the merit function, M, is calculated for each individual as above. See APPENDIX A: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A GRADIENT-INDEX SHAPER for an example of a CODE V macro

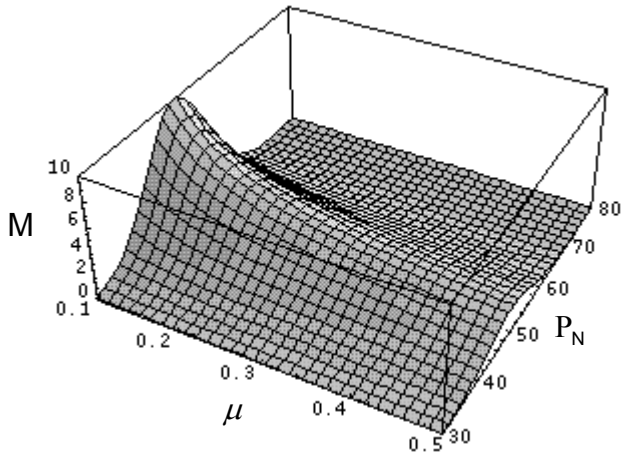

Fig. 8. Merit function versus μ and P_N. $N = 200$ in this system (from Ref. 14).

used to evaluate the merit function and call the GA library. The constraints on each of the surface-shape parameters are given in Table 1.

Table 1. Constraints on surface parameters. Each parameter must be between or equal to the end points of the respective constraint.

Surface Parameter	Constraint
c	-10 to 20
k	-1.0 to 0
A_4	-1.0×10^{-3} to 1.0×10^{3}
A_6	-1.0×10^{-5} to 1.0×10^{5}
A_8	-1.0×10^{-6} to 1.0×10^{6}
A_{10}	-1.0×10^{-7} to 1.0×10^{7}

The GA code was executed simultaneously on four Sun Sparcs, all running Solaris 2.6. The constraints were modified in real-time so that each instantiation of the GA code could search different regimes of the parameter space. Once it became apparent that a particular regime contained better solutions, the constraints were narrowed on all machines to search that regime more thoroughly. The constraints given in Table 1 represent the final values of these constraints. Also, when one machine found an individual that was substantially superior to the best individuals on the other three machines, the code on the three other machines was re-initialized using a restart file from the machine with the superior individual. This amounts to a primitive form of parallel processing, an issue which will be better addressed in future applications (see Parallelization of the Genetic Algorithm). Total processing time was not rigorously recorded but was on the order of 12 hours. The fastest machine of the four, a Sun Ultra 1 170 with 64M of RAM, found the best individual. The search was stopped when no

significantly better individuals were found over a period of five hours, so the best individual actually was discovered after about 7 hours of pro N cessing time.

The final profile-shaping system is shown in Fig. 7, with the GA-determined shaping element shown in Fig. 9. In Fig. 7, one notices that the GA-determined shaping element is actually the second element in the system; the fi $\bar{u}$ rst 3.78×10^{-5} rays/mm^2 element (the 'thin lens element') is an artifact of the initial design requirements and is not part of the optimization process. The purpose of the thin lens element is to focus the incoming collimated beam such that the Numerical Aperture of the shaping element is 0.7. The two surfaces of the thin lens element are spherical and the shape factor is set such that spherical aberration is minimized.[58] The shape of the input beam profile is shown in Fig. 10. It is assumed that the laser beam is circular and is operating in the fundamental mode, TEM_{00}. In Fig. 11, one can see that irradiance on the Output Surface is nearly uniform, with a average value, $\bar{u}$, of 2.13×10^{-3} rays/mm^2. To check for self-consistency, the irradiance functions, $\sigma(\rho)$ and $u(\mathrm{P})$, are integrated over the Input Plane and Output Surface, respectively. The results of these integrations are in close agreement, as expected. The uniformity of the output profile can be characterized by the standard deviation from the mean value of the points that determine the output profile. The standard deviation for this data set is , which is 1.8% of . The general parameters for the system are given in Table 2, along with specific parameters for the shaping element and thin lens element in Table 3.

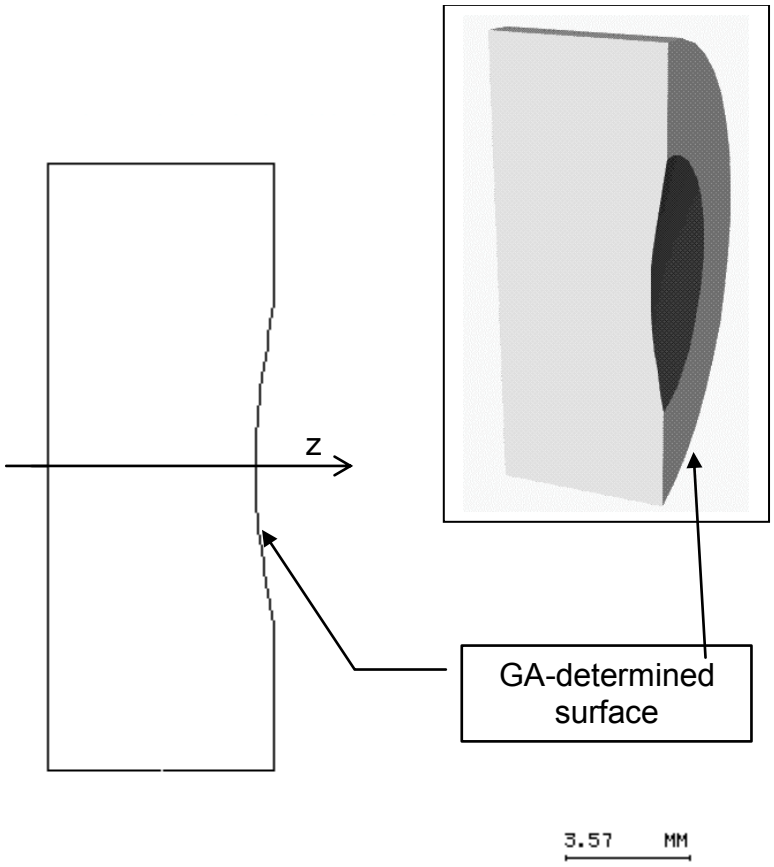

Fig. 9. Shaping element showing aspherical surface, which is determined by the GA. The axial thickness of this element is 6 mm (from Ref. 14).

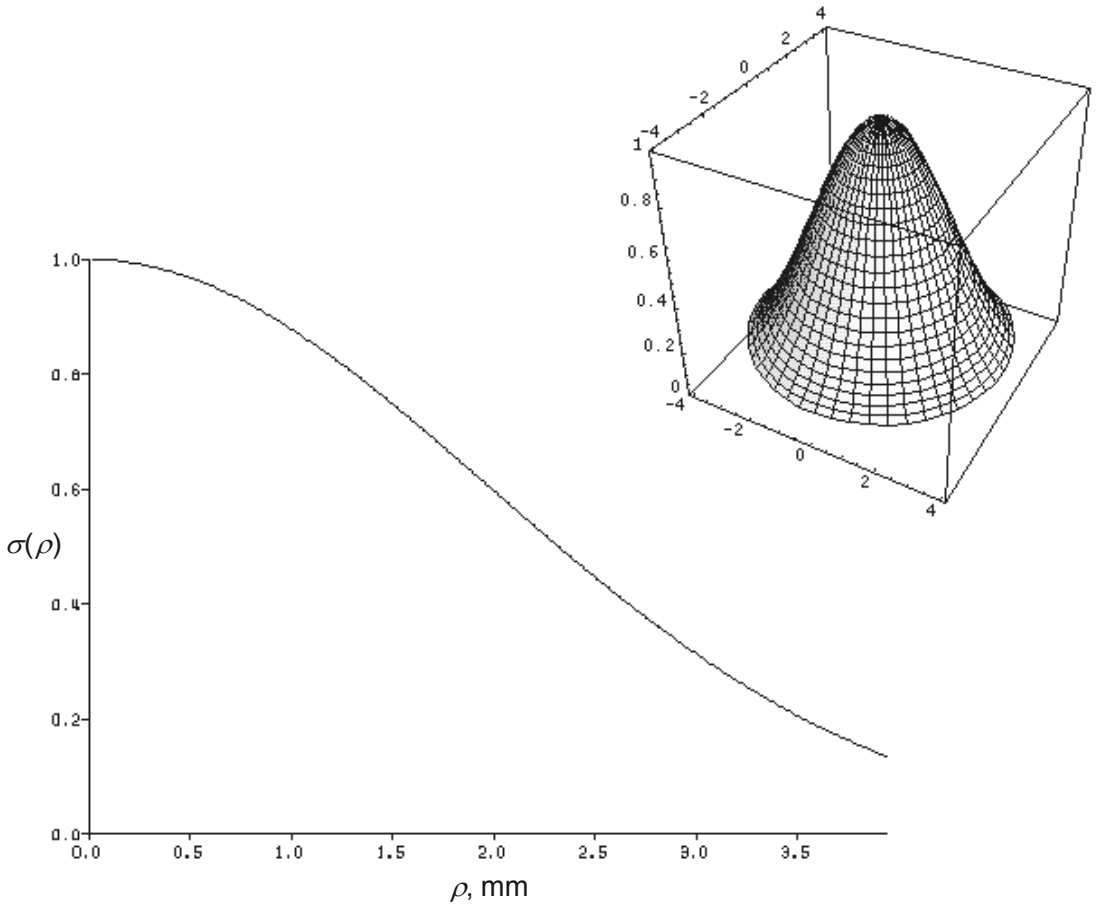

Fig. 10. Input beam irradiance profile. The $1/e^2$ diameter of the input beam is 7.882 mm. Integrating $\sigma(\rho)$ over the Input Plane yields 21.1 units, a quantity which must be conserved according to Eq. (13) (from Ref. 14).

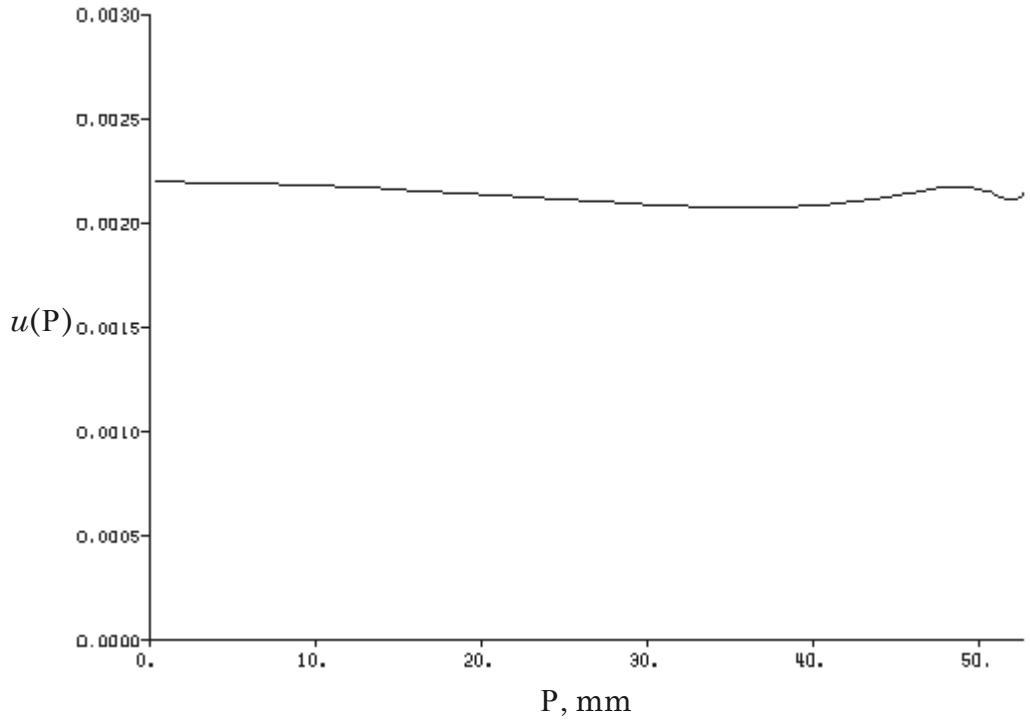

Fig. 11. Beam profile on Output Surface. The radius (P_N) of the Output Surface is 52.5 mm. The mean value of the profile, $\bar{u}$, is 2.13×10^{-3} rays/mm^2, with a standard deviation of 3.78×10^{-5}. Integrating this mean value ($u(P) = \text{constant} = \bar{u}$) over the Output Surface yields a value of 20.7 units (from Ref. 14). The beam profile is radially-symmetric.

Table 2. Beam Shaper/Projector System Parameters.

Parameter	Value
Wavelength	441.57 nm
Radius of the input beam (Entrance Pupil Diameter)	3.9441 mm
Radius of the output aperture	52.5 mm
Glass type for two lens elements	Ohara slah53
Index of ambient medium (air)	1.0
Gaussian constant α in Eq. (21)	0.129 mm^{-2}
Object distance	Infinity
Image distance from shaping surface	242.9 mm

Table 3. Beam Shaper/Projector Lens Element Parameters

	Thin Lens Element		Shaping Element	
Parameter	Left Surface	Right Surface	Left Surface	Right Surface
Aperture Radius			8.0 mm	8.0 mm
Thickness	1.0 mm	49.1 mm	6.0 mm	242.9 mm
Vertex radius ($1/c$)	36.606 mm	359.72 mm	infinity	14.235 mm
Surface type	spherical	spherical	spherical	aspherical
Conic constant (k)				0.0
A_4				-0.66411×10^{-3} mm
A_6				0.12400×10^{-5} mm
A_8				-0.31156×10^{-6} mm
A_{10}				-0.61495×10^{-8} mm

The GA method produced the desired system--that is, a system with a uniform irradiance profile on the spherical Output Surface and with an exit pupil very close to 50 mm--in a reasonably efficient manner and while requiring virtually no user input. The GA started with a randomly defined system and found a good solution in about 7 hours. The self-consistency check, which involves integrating the input irradiance

function, $\sigma(\rho)$, over the Input Plane and integrating the output irradiance function, $u(\mathrm{P})$, over the Output Surface, indicates energy is conserved as required by Eq. (13). The small difference in the two values (1.9% error) can be attributed the small deviations from the mean ($\overline{u}$) in the output irradiance profile data set, as shown in the previous paragraph. Nevertheless, this error would certainly fall within an acceptable range of fabrication for an aspherical surface such as those in this problem.[59]

This application illustrates the ability of GAs to solve difficult problems, suggesting the GA may be useful in solving more complex and vexing problems in optics. The efficiency of the GA method can be enhanced by introducing parallelism into the GA code. The most computationally expensive step in the GA routine is the calculation of the merit function, which requires N rays to be traced through the system for each individual in a generation. This is done in serial and no other part of the code can execute until the merit function has been calculated. If merit functions for various individuals are calculated in parallel on several slave machines simultaneously, allowing the GA to run unfettered on a master machine, the efficiency of the overall search process will be enhanced significantly.

Design and Analysis of a Two-lens Beam Shaper

This system represents the most complex presented thusfar, where complexity is related to the dimensionality of the merit-function space. This problem is inspired by the system designed, built and tested by Jiang, Shealy and Martin in Ref. 16. The system has two lens elements, which are designed to shape an incoming Gaussian beam to an outgoing beam with a uniform irradiance profile. The system expands an 8mm incoming beam to 12mm. Both incoming and outgoing beams are parallel to the optical axis. The right surface of the first lens and the left surface of the second lens

accomplish the irradiance redistribution and beam expansion. In Ref. 16, these shaping surfaces are highly asperical.

As above, the merit function must contain a term that quantifies the uniformity of the irradiance profile on an Output Surface. Furthermore, the merit function must insure that the outgoing rays are parallel to the optical axis and that the radius of the output beam is some predefined value. The precise nature of this merit function is similar to those described in the previous problems. The GA is given 12 parameters to optimize in this problem: $c, k, A_4, A_6, A_8,$ and A_{10} for the aspherical lens surface of each of the two lens in the system. The merit function for this system has the same form as the one for the Beam Shaper/Projector system outlined in Design and Analysis of a Beam Shaper/Projector [Eqs. (23)-(25)], save that '50' in Eq. (23) is '12' in this example (since the radial height of the marginal ray should be 12 mm in this example and not 50 mm). This 12-dimension parameter space is the most complex presented thus far. Consequently, the optimization time for this problem is significantly longer, taking some 50 hours on the platforms described above, using the serial processing paradigm. The GA, nevertheless, found a solution, which is presented in Fig. 12, Fig. 13, Table 4, and Table 5.

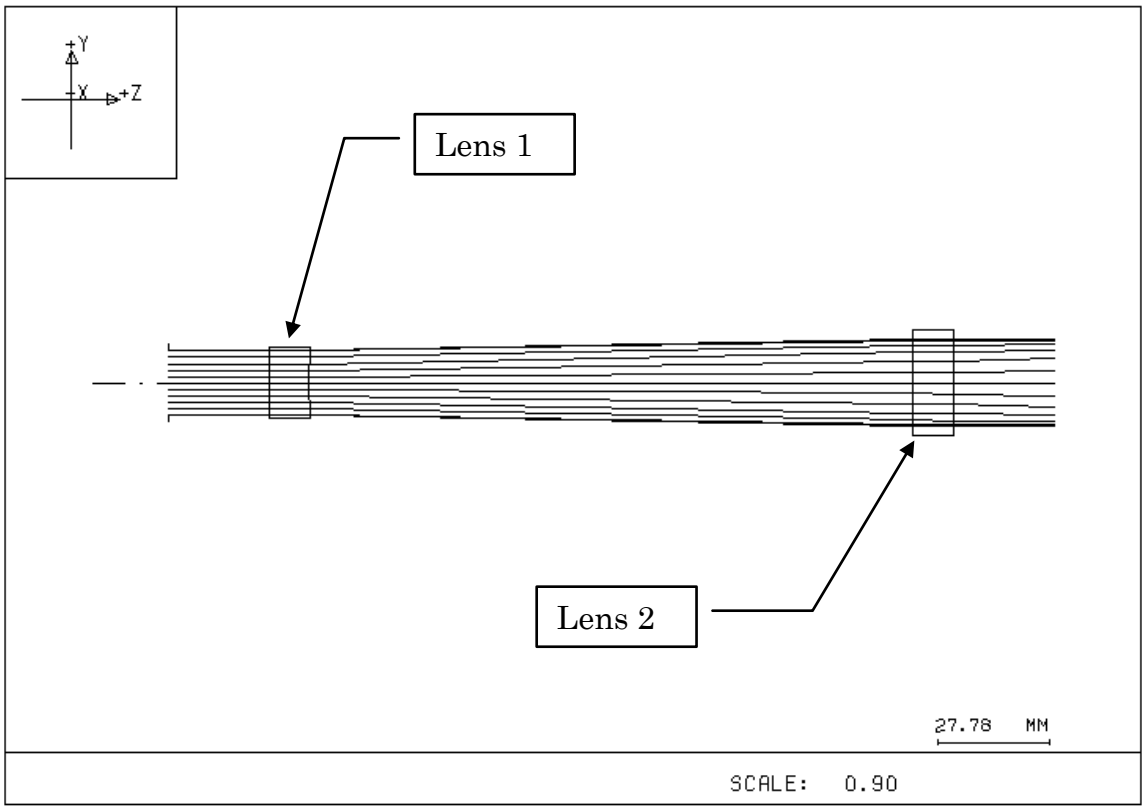

Fig. 12. Two-lens beam shaper system with ray trace. The right surface of Lens 1 and the left surface of Len 2 are shaped by the GA.

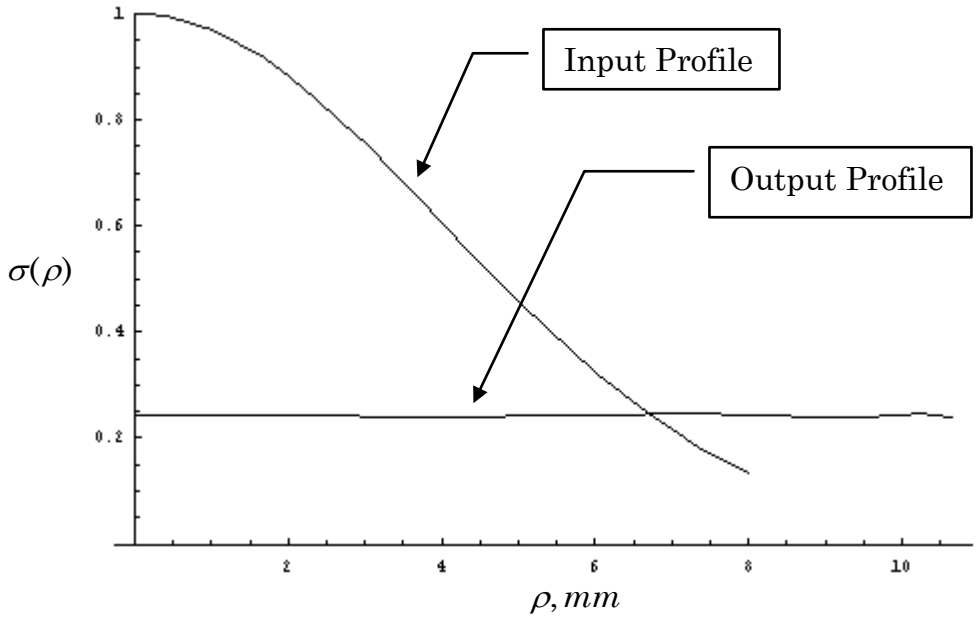

Fig. 13. Input and Output irradiance profiles for the Two-lens Beam Shaper. The $1/e^2$ diameter of the input beam is 16.0 mm. Integrating $\sigma(\rho)$ over the Input Plane yields 86.9 units. The radius (P_N) of the Output Plane is 10.7 mm. The mean value of the profile, $\bar{u}$, is 0.242 rays/mm^2, with a standard deviation of 1.86x10^{-3} rays/mm^2, or 0.8% of $\bar{u}$. Integrating this mean value ($u(\text{P}) = \text{constant} = \bar{u}$) over the Output Surface yields a value of 87.0 units. Energy is conserved, as required in Eq. (13).

Table 4. Two-lens Shaper System Parameters.

Parameter	Value
Wavelength	589.00 nm
Radius of the input beam (Entrance Pupil Diameter)	8.00 mm
Radius of the output aperture	10.7 mm
Glass type for two lens elements	CaF_2
Index of ambient medium (air)	1.0
Gaussian constant α in Eq. (21)	0.031 mm^{-2}
Object distance	Infinity

Table 5. Two-lens Shaper Lens Element Parameters

	First Lens Element		Second Element	
Parameter	Left Surface	Right Surface	Left Surface	Right Surface
Thickness	10 mm	150 mm	10 mm	25 mm
Vertex radius ($1/c$)	infinity	100.59 mm	-100.01 mm	infinity
Surface type	spherical	aspherical	aspherical	spherical
Conic constant (k)		-0.922971	-0.375469	
A_4		0.843226x10^{-5} mm	0.617298x10^{-4} mm	
A_6		-.664541x10^{-6} mm	-.960417x10^{-7} mm	
A_8		0.504624x10^{-8} mm	-.164098x10^{-8} mm	
A_{10}		0.274667x10^{-14} mm	0.687429x10^{-11} mm	
A_{12}		-.999878x10^{-13} mm	0.466018x10^{-14} mm	

Design and Analysis of a Gradient-Index Shaper

The Beam Shaper/Projector problem presented in the previous section provides an example of a purely continuous merit function. Continuous merit functions can be solved by a number of different methods, and solving it with a GA is nothing particularly glamorous. As an example of a more complex problem—one difficult to

solve using more conventional methods—the GA technique is used to design a Gradient-Index Shaper, which has a merit function that contain both continuous and discrete parameters. This problem was solved using a differential-equation design method in a paper by Wang and Shealy.[17] Though Wang was able to produce several perfectly good solutions, the gradients of the lenses reported in Ref. 17 were determined solely by solving the differential equations, and no constraints were imposed that required the indices of refraction of the lenses to correspond to those that can be found in common glass catalogs. It follows that a challenging problem for the GA would be to create a laser shaping system with two gradient-index elements as Wang did, but to do so with the added constraint that the elements can only be chosen from existing glass catalogs. Obviously, this makes the potential for fabricating the system the more easily realized. Furthermore, this is an interesting problem from the perspective of the GA since it is now required to optimize discrete parameters in addition to continuous parameters, adding several nuances to the coding.

The general layout of the gradient-index shaping system is inspired by Wang's system. Essentially, there are two shaping elements, each of which are made from a gradient-index glass. The right surface of each shaping element is spherical. The precise shape of the spherical surface and the thickness for each shaping element are optimized by the GA. Furthermore, there is a 'connector' between the two shaping elements that is set a priori to be Schott BK7 glass type. The 'connector' in Wang's system matches the base index, n_0, of the two gradient elements (see Fig. 14). This is necessary to insure that the marginal ray passes undeviated through the system.[17] No such requirement is made for the GA problem, so the glass type of the connector can be set with caprice. The GA is allowed to choose randomly from the Lightpath gradient-

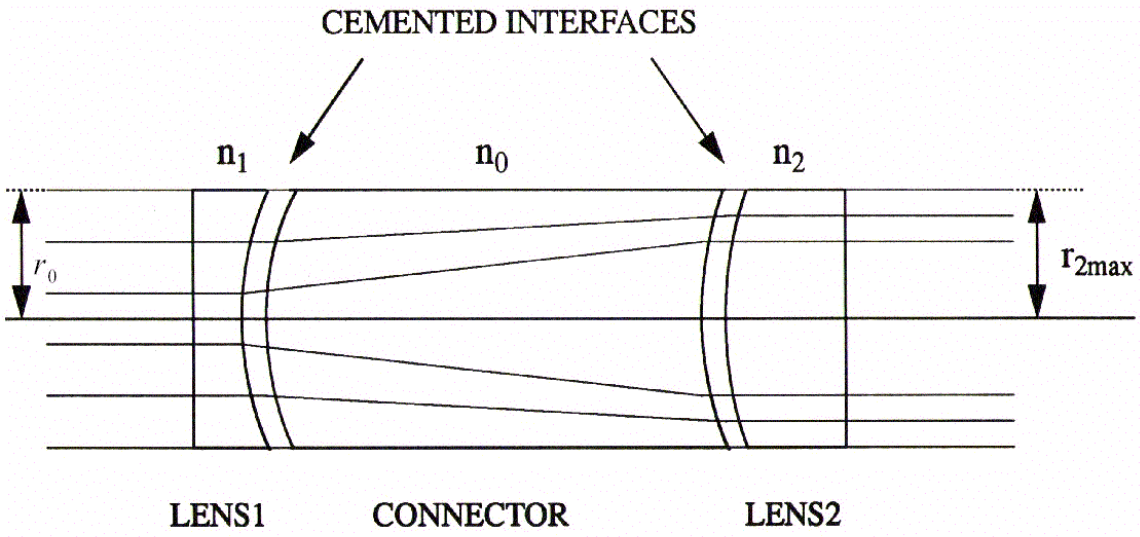

Fig. 14. Layout of the gradient-index expander designed by Wang and Shealy (from Ref. 17). This system provides the inspiration for the Gradient-Index Shaper problem.

index glass catalog available in CODE V. See CODE V documentation for more details on this glass catalog. Thus, the GA optimizes the following continuous parameters: thickness of element 1, curvature of right surface of surface 1, thickness of connector, thickness of element 2 and curvature of right surface of surface 2. Also, the GA optimizes the following discrete parameters, Lightpath gradient glass type for surfaces 1 and 2 (four possible types for each surface) and gradient index direction for surface 1 and 2 (may be positive or negative for each respective surface). While the problems presented thus far have a continuous parameter space, this problem does not. Because of this, derivative-based methods (e.g. simplex and Damped Least Squares[60]) are not applicable.

The merit function for this system is designed with three key features. First, the merit function includes a term to characterizes the flatness of the output profile, in the manner as shown in Design and Analysis of a Beam Shaper/Projector. Second, the merit function includes a term the insures that each ray is perpendicular to the Output Plane, i.e., a collimated output beam. The form of this term is expressed by the following function:

$$\exp\left[-(1-\Theta)^2\right], \tag{26}$$

where

$$\Theta = \prod_{i=1}^{N} \gamma_i^s \ . \tag{27}$$

γ_i^s in the above equation is the cosine of angle that ray i makes with the optical axis. The exponent s adjusts the sensitivity of the merit function non-parallel ($\gamma \neq 1$) rays. For this application, s is set at six. If each ray traces through the system is parallel to the optical axis, then $\Theta = 1$ and Eq. (26) is unity. If not, then Eq. (26) is less than one and reduces the merit function, penalizing the system.

Third, the merit function rewards those systems where the radius (P_N) of the Output Plane is close to 8.0 mm. This is expressed in the following function:

$$\exp\left[-0.01\,(8-\mathrm{P}_N)^2\right], \tag{28}$$

where P_N is the radial height of the marginal ray at the Output Surface. If P_N is 8.0 mm as desired, then Eq. (28) is unity. If not, then the system is penalized as above. See APPENDIX A: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A GRADIENT-INDEX SHAPER, line 101 to see these terms as they appear in the code, expressed as a complete function. Descriptions of the terms in line 101 are given at the end of Appendix A. Building the merit function in this manner is intended to produce a system that resembles the systems in Ref. 17, which makes similarities and differences more apparent.

To solve this problem, the serial version of the GA is employed and CODE V calculates the merit function. After a total execution time of about 7 hours, the highest value of the merit function (the best individual) failed to significantly improve indicating no better solutions were forthcoming. The final system is shown in Fig. 15 and the shapes of the input and output irradiance profiles, along with consistency checks, are shown in Fig. 16. Examining this best individual, one finds that the integrating the irradiance functions, $\sigma(\rho)$ and $u(\mathrm{P})$ over the appropriate surfaces

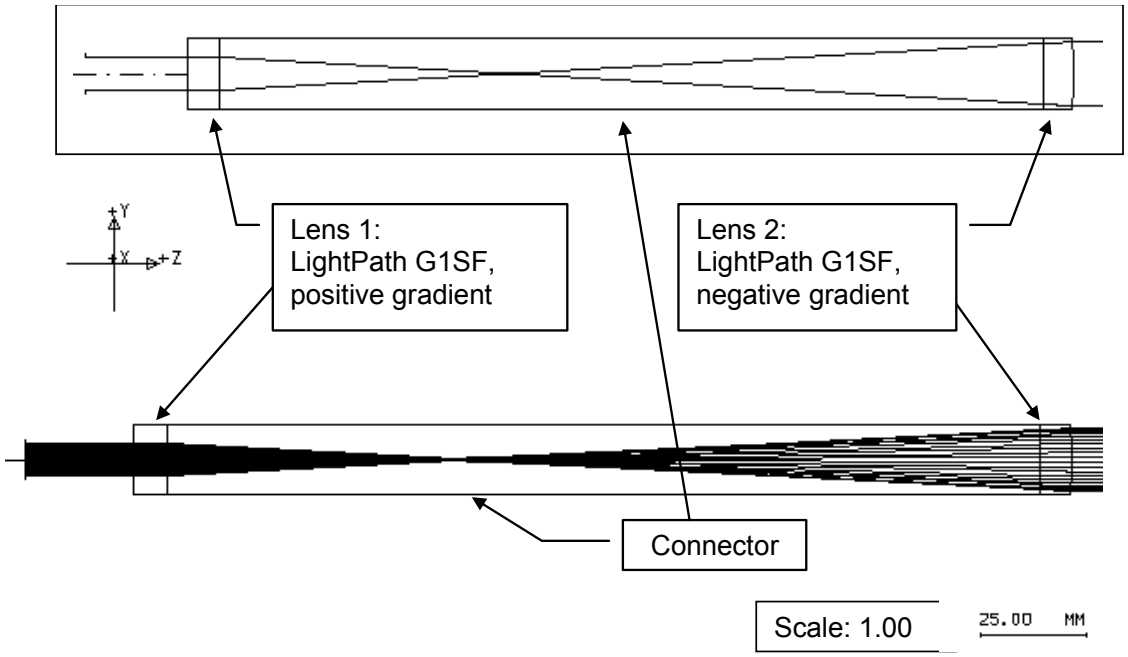

Fig. 15. Gradient-index shaper system with ray trace. The materials for Lens 1 and Lens 2 were chosen from a catalog of materials by the GA.

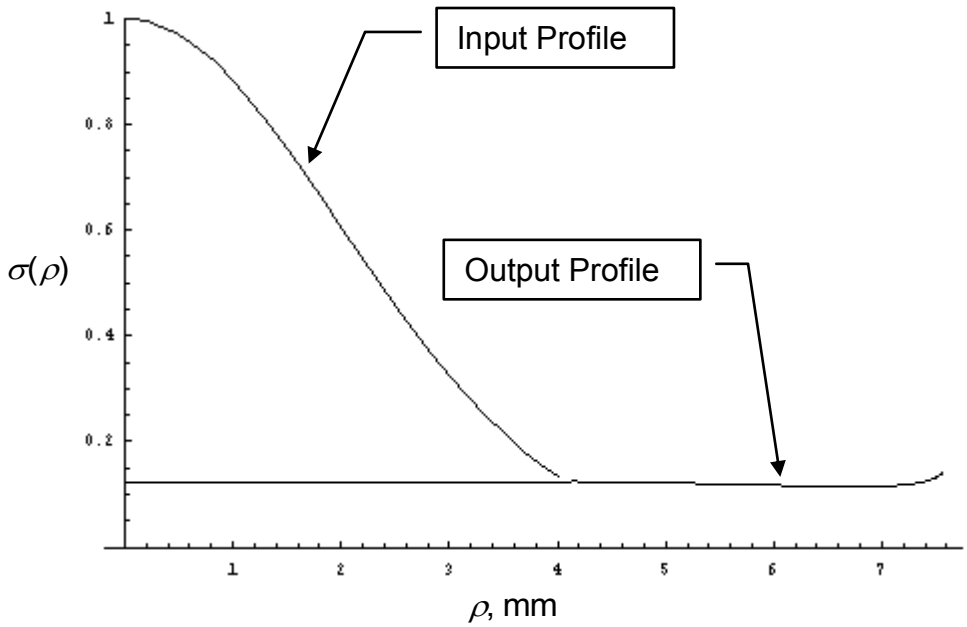

Fig. 16. Input and Output irradiance profiles for the Gradient-Index Shaper. The $1/e^2$ radius of the input beam is 4.0 mm. Integrating $\sigma(\rho)$ over the Input Plane yields 21.7 units. The radius (P_N) of the Output Plane is 7.56 mm. The mean value of the profile, $\bar{u}$, is 0.121 rays/mm^2, with a standard deviation of $4.45\text{x}10^{-3}$. Integrating this mean value ($u(\text{P}) = \text{constant} = \bar{u}$) over the Output Surface yields a value of 21.7 units. Energy is conserved, as required in Eq. (13).

yields the same value. Energy is conserved as expected. The system satisfies all given constraints and performs with the necessary features; thus, the GA has indeed solved the problem. The parameters for the final system are shown in Table 6 and Table 7. It is interesting to note that the marginal rays converge inside the connector, resulting in a very long connector length. Such a long connector would be undesirable in any system intended for manufacture. By referring to Wang's system, one notes that the length of the connector is significantly smaller than the one produced by the GA. This suggests that adding an additional constraint which limits the length of the connector to some small value could force the GA to produce a system with collimated marginal rays. The index of refraction of the connector would also need to be a variable to satisfy the physical constraint that the marginal ray, defined as the ray that enters at the $1/e^2$ point ($\rho_N = 4.0\ \text{mm}$ in this case), passes undeviated through the system.

Table 6. Gradient-Index Shaper System Parameters.

Parameter	Value
Wavelength	589.00 nm
Radius of the input beam (Entrance Pupil Diameter)	4.00 mm
Radius of the output aperture	7.56 mm
Glass type for two lens elements	Lightpath G1SF
Gradient for first lens element	positive
Gradient for second lens element	negative
Glass type for connector	Schott BK1
Index of ambient medium (air)	1.0
Gaussian constant α in Eq. (21)	$0.035\ \text{mm}^{-2}$
Object distance	Infinity

Table 7. Gradient-Index Shaper Lens Element Parameters

Parameter	First Lens Element		Second Element	
	Left Surface	Right Surface	Left Surface	Right Surface
Thickness	7.75 mm	199.9 mm	7.29 mm	25.0 mm
Vertex radius ($1/c$)	infinity	-99.58 mm	infinity	-58.15 mm
Surface type	spherical	spherical	spherical	spherical

Design and Analysis of a Free-Form GA-Designed GRIN Shaper

While the previous examples establish the GA's ability to arrive at solutions for several well-understood problems, they do not fully harness the *creative* ability of the GA method. If the GA were given more flexibility, from choosing the actual number and types of GRIN elements, rather than just their shapes, could the GA actually produce a solution in a reasonable amount of time? In searching for a problem to address this question, one notes that an encumbrance of the GRIN system (based on one developed by Wang and Shealy[17]) presented earlier in this work is that it requires a 'connector' piece. In the Wang and Shealy system (see Fig. 14), this connector allows the marginal ray to trace parallel to optical axis. The solution found by the GA presented in this work has all rays focusing within the connector (see Fig. 15), which is an undesirable feature from a fabrication perspective (the connector may be damaged if a high-power laser is used). Since this possibility was never considered beforehand, no term in the merit function was added to penalize solutions where marginal rays focus within the connector. Though not by design, this illustrates the creative nature of the GA, in the sense that the GA may find solutions to a given problem that the human designer never imagined.

Thus, a new problem may be defined: is there a system that shapes an input beam to a flat profile in manner similar to that of the original Wang and Shealy GRIN system, but without requiring a connector piece between the lens elements whatsoever? That is, is there a shaping system that uses GRIN elements separated only by ambient air? Furthermore, if the GA is no longer limited by specifying so many aspects of the optical system *a priori*, such as the actual number of elements present in the system, can the GA harness this greater flexibility in determining the makeup of the optical system to solve the problem? To do this, the GA not only must optimize surface shapes of the GRIN elements and their spacing, but also must determine the actual number of elements in the solution, up to a certain limit (four, in this case). In essence, to solve this problem, the GA must do the same things that a human optical designer would. The number of GRIN elements must be determined, the type of GRIN material for each element must be picked, and so on. This new type of problem further distinguishes the GA method from deterministic methods (i.e., those that rely on derivatives and a smooth, continuous merit function) since the merit function required for this problem is a complicated mix of discrete and continuous parameters.

The merit function is constructed in a manner similar to the previous problems. It contains three key terms: one term that measures the uniformity of the output profile, one term that insures that all rays exit the system parallel to the optical axis and finally, one term that penalizes systems that do not have the specified radius on the Output Surface (4 mm, in this case). One unique aspect of this application is the way that the GA is allowed is select the actual number of elements for each test system. This variable, N_{ele}, may be one of the following integers: one, two, three or four. An idiosyncrasy of the particular GA code[47] used in these examples is that parameters to be

optimized are defined by default as 8-bit real numbers. The parameter in the code that corresponds to N_{ele} (an integer) is $P(i,1)$ (an 8-bit real number), where i refers to the one of the ten individuals in a generation. For this problem, there are actually 26 parameters to be optimized, and the population size of each generation is 10. Thus, P is a array with 10 rows and 26 columns. In order to translate $P(i,1)$ into an integer within the desired range, the limits are set as follows:

$$0.50 \le P(i,1) \le 4.49; \tag{29}$$

and, the following function is used for translation:

$$N_{ele}(i) = \text{int}\left[P(i,1)\right], \tag{30}$$

where $\text{int}(a)$ is a function that rounds a to the closest integer. Using Eqs. (29) and (30) , an integer between and including one and four is chosen, each with equal probability.

It is this method that is used to define discrete parameters for the GA in all these problems that require them. In this problem, there are eight other discrete parameters: the GRIN glass types and the GRIN direction for each of the (up to) four elements. These glass types are selected from a catalog of GRIN elements for a particular manufacturer in the same manner as in the previous example. See lines 34 and 40 in APPENDIX A: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A GRADIENT-INDEX SHAPER for an example of this in the code. See Table 8 for a description of all parameters optimized in this problem. It should be noted that if the GA chooses less than four elements for a particular individual, the remaining surfaces are made into dummy surface in the code. For example, if $P(i,1) = 3$, then the following occurs: first, the left and right surface of element four is set to have an infinite radius of curvature, i.e. the surfaces are made flat. Then, the thickness of element four is set to

zero and the glass type is set to air. The net result is that element four has no effect on the optical system, while the surfaces for element four remained defined for future use, should they become necessary. This method simplifies coding of the problem and makes evaluation of the merit function more efficient. See APPENDIX B: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A FREE-FORM GA-DESIGNED GRIN SHAPER for example of this in code.

The code was executed on a Sun Ultra 1 170 with 64M of RAM as before. Using this setup, it took an average of 7.80 seconds for the GA to completely evaluate the merit function for each of the 10 test systems (individuals) in a generation. The code was allowed to run until it reached generation number 12,367, resulting in an effective total run-time of 26.8 hours. The merit function, which is measured in arbitrary units, peaked at a value of $M = 101.21$. The fact that this value represents the best solution to the problem can be seen in Fig. 17, where it is clear that M_{best} asymptotically approaches a value of about 102.

The system with a merit function value of $M = 101.21$ is represented in Fig. 18. The system has 3 elements, all with spherical surfaces. Most importantly, this solution reshapes the Gaussian Input Profile into a irradiance profile with uniform intensity on the Output Surface. One finds that integrating the Irradiance Profile over the Output Surface yields 21.9 units, while integrating the specified Irradiance Profile over the Input Surface yields 21.7 units. As with the Beam Shaper/Projector system, this error can be attributed to the small deviations from $\bar{u}$ in the output irradiance profile data set. Here, $\bar{u}$ is the mean of $u(\mathrm{P})$, which is defined in Eq. (20). The Input and Output profiles are shown in Fig. 19. The parameters for the systems are given in Table 9 and Table 10.

Table 8. Optimized Parameters for the Free-Form GA-Designed GRIN Shaper problem.

Para-meter	Parameter Description	Parameter Type	Parameter Limits61
1	Number of Elements	Discrete	1-4 (integer)
2	Radius of curvature of left surface of Element 1	Continuous	-100 to 100
3	Radius of curvature of right surface of Element 1	Continuous	-100 to 100
4	Thickness of Element 1	Continuous	1 to 10
5	Distance between Element 1 and Element 2	Continuous	1 to 10
6	GRIN glass type for Element 1	Discrete	1-6 (integer)
7	Positive or Negative GRIN for Element 1	Discrete	0, negative or 1, positive
8	Radius of curvature of left surface of Element 2	Continuous	-100 to 100
9	Radius of curvature of right surface of Element 2	Continuous	-100 to 100
10	Thickness of Element 2	Continuous	1 to 10
11	Distance between Element 2 and Element 3	Continuous	1 to 10
12	GRIN glass type for Element 2	Discrete	1-6 (integer)
13	Positive or Negative GRIN for Element 2	Discrete	0, negative or 1, positive
14	Radius of curvature of left surface of Element 3	Continuous	-100 to 100
15	Radius of curvature of right surface of Element 3	Continuous	-100 to 100
16	Thickness of Element 3	Continuous	1 to 10
17	Distance between Element 3 and Element 4	Continuous	1 to 10
18	GRIN glass type for Element 3	Discrete	1-6 (integer)
19	Positive or Negative GRIN for Element 3	Discrete	0, negative or 1, positive
20	Radius of curvature of left surface of Element 4	Continuous	-100 to 100
21	Radius of curvature of right surface of Element 4	Continuous	-100 to 100
22	Thickness of Element 4	Continuous	1 to 10
23	Distance between Element 4 and Surface 10 (a dummy surface)	Continuous	1 to 10
24	GRIN glass type for Element 4	Discrete	1-6 (integer)
25	Positive or Negative GRIN for Element 4	Discrete	0, negative or 1, positive
26	Distance from Surface 10 (a dummy surface) to the Output Plane	Continuous	1 to 100

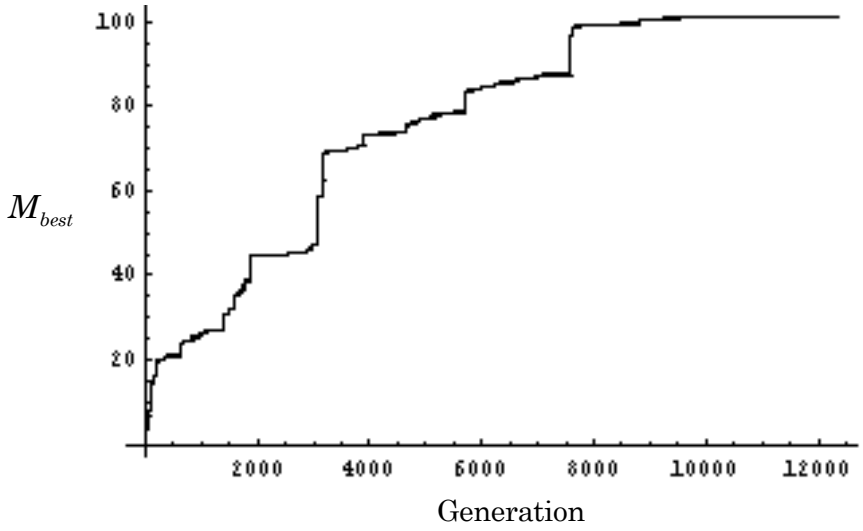

Fig. 17. A plot showing the best individual in a generation, M_{best}, as a function of generation. M_{best} is measured in arbitrary units. When M_{best} 'plateaus' for a significant number of generations, it can be assumed that the best solution has been found.

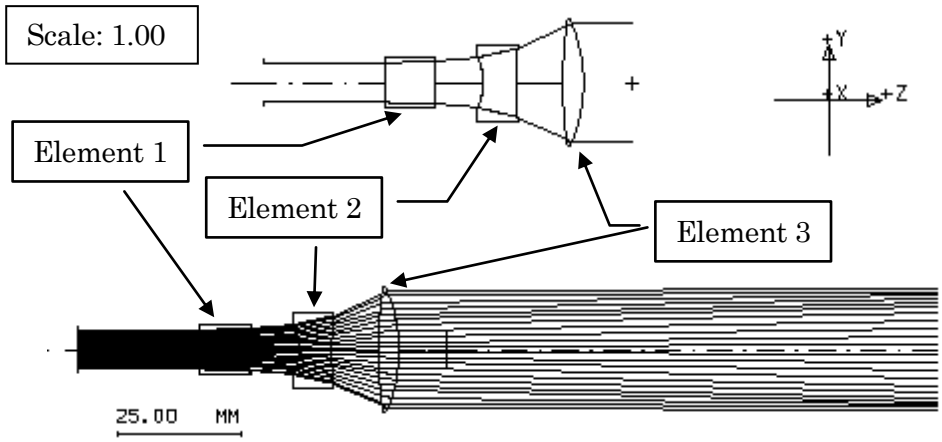

Fig. 18. Raytrace for the Free-Form GA-Designed GRIN Shaper system. The GA produced a system with 3 elements and no connectors.

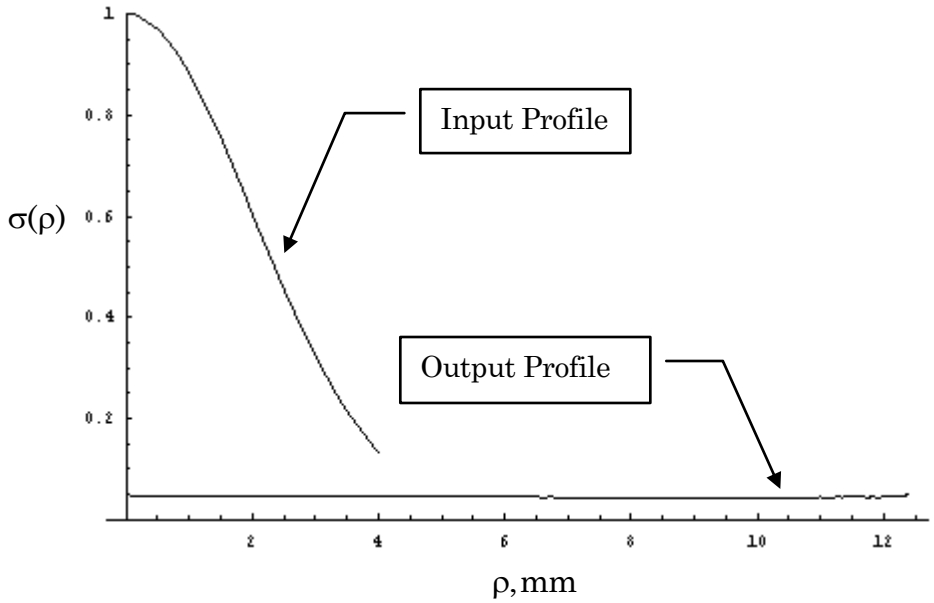

Fig. 19. Input and Output irradiance profiles for the Free-Form GA-Designed GRIN Shaper. The $1/e^2$ radius of the input beam is 4.0 mm. Integrating $\sigma(\rho)$ over the Input Plane yields 21.7 units. The radius (P_N) of the Output Plane is 12.4 mm. The mean value of the profile, $\bar{u}$, is 4.55×10^{-2} rays/mm^2, with a standard deviation of 1.70×10^{-3}, or 3.7% of $\bar{u}$. Integrating this mean value ($u(P) = \text{constant} = \bar{u}$) over the Output Surface yields a value of 21.9 units. Energy is conserved as required in Eq. (13).

Table 9. Free-Form GA-Designed GRIN Shaper Parameters.

Parameter	Value
Wavelength	589.00 nm
Radius of the input beam (Entrance Pupil Diameter)	4.00 mm
Radius of the output aperture	12.4 mm
Index of ambient medium (air)	1.0
Gaussian constant α in Eq. (21)	0.035 mm^{-2}
Number of Elements	3
Distance from Surface 10 to Output Place (parameter 26 in Table 8)	100 mm
Object distance	Infinity

Table 10. Free-Form GA-Designed GRIN Shaper Lens Parameters.

Parameter	First Element		Second Element		Third Element	
	Left Surface	Right Surface	Left Surface	Right Surface	Left Surface	Right Surface
Thickness, mm	9.99	10.0	6.77	9.48	4.25	9.71
Vertex radius ($1/c$), mm	-61.6	80.4	-12.5	100	87.5	-30.3
Surface type	spherical	spherical	spherical	spherical	spherical	spherical
Glass Type (UDG C1)[62]	3		2		1	
GRIN Direction	negative		positive		negative	

To review, the GA method produced a shaper system that only has spherical surfaces, requires no connectors and uses GRIN lenses from the catalog of an established GRIN element manufacturer. The GA solved a problem that would be difficult to solve using analytical methods or conventional optimization techniques, since the merit function contains discrete parameters (for example, picking the GRIN glass type from a pre-defined set of GRIN elements). Furthermore, the GA was presented with a unique problem that has not been solved before and was allowed a certain degree of creativity in producing a solution. The result is a three-lens system

that is wholly the creation of the GA and has a form that could not have been anticipated before the fact.

SUMMARY AND CONCLUSIONS

This work starts with the following assumption: there are some problems in geometrical optics that cannot be solved using either analytical methods, such as the differential equation method[11,13], or conventional computational optimization techniques, such as the Simplex algorithm.[37] There are numerous reasons that account for this, ranging from merit functions that contain non-continuous parameters to problems that simply have no solution. To this end, the following hypothesis is proposed: special machine learning codes--in particular, genetic algorithms--can be adapted to solve many of these problems. The first step in testing this hypothesis is to establish that GAs can indeed produce solutions to problems in geometrical optics. This is done by addressing a few well-understood problems from the literature, which have been solved by other methods. In addition to establishing a proving ground, these exercises shed light on the myriad subtle qualities that govern the GA method. Hopefully, this information can be used to make the GA code run more efficiently and to determine the limits of its application.

The first two problems presented here, Design and Analysis of a Beam Shaper/Projector and Design and Analysis of a Two-lens Beam Shaper, both could be solved by more conventional methods. In fact, the Two-lens Beam Shaper problem has been solved by Jiang and Shealy via an analytical differential equation method.[16] The GA method indeed produced solutions to these problems, and did so within a reasonable amount of time. The solution to the Two-lens shaper problem produced by the GA is

very similar to the one found by Jiang and Shealy. Yet, both of these solutions, in addition to the solution to the Beam Shaper/Projector problem, require elements with highly aspherical surfaces. Aspherical surfaces are used in these problems since often exotic surfaces are required to produce the desired solutions. This greater flexibility makes finding a solution easier, both for the GA method and analytical methods. However, choosing aspherical surfaces comes at a price; they are both expensive to manufacture and difficult to build to precise specifications.

With this in mind, Wang and Shealy endeavored to produce a beam shaping system similar to Jiang's Two-beam shaper but with the additional constraint that only the solution contains only spherical elements. To compensate for the decreased flexibility as a result of requiring spherical surface shapes, gradient-index glasses were chosen for the shaping elements. Using a differential equation design method, Wang and Shealy produced the desired shaping system.[17] While the system performs its primary function of shaping a Gaussian input beam into a uniform profile on the Output Surface, it has two undesirable features. First, a connector was added to allow the marginal ray to pass through the system unfettered by any of the shaping elements. This requirement is necessary to insure the that radius of the input beam is equal to that of the output beam. Also, the parameters defining the index of refraction function for each of the GRIN elements in Wang's system were allowed to vary continuously, resulting in GRIN elements that could not be matched with elements found in established GRIN manufacturer catalogs. That is, Wang's GRIN elements must be custom fabricated, adding significantly to the cost required to build that system.

To address this issue, the GA was given the task of producing a system similar to Wang's except that the GRIN elements were to be chosen from a predefined set. This set is simply a catalog of elements from a popular GRIN manufacturer (Lightpath was

used in this case, though any catalog would have been sufficient). With this constraint, the problem contains parameters that can only assume certain values, and hence do not vary smoothly. This aspect makes this problem difficult to solve with methods that rely on continuous merit functions, which eliminates the differential equation method used by Wang. This is the first problem where the GA was used to produce a truly unique solution to a difficult problem. The system found by the GA contains only spherical surfaces, has GRIN elements chosen by the GA from a catalog and shapes the Gaussian input beam into one with a uniform irradiance profile on the output surface, as desired. However, it also has one unanticipated feature. The marginal rays actually focus within the connector (see Fig. 15). New terms might be added to the merit function to insure that rays do not focus within the connector, and the entire optimization process restarted to find a new solution. Rather than follow this course, however, one finds that the accidental discovery of this feature shows the way to a new, more ambitious problem: the total elimination of the connector.

In the Free-form GA-designed shaping system, the last problem presented in this work, the goal was to produce a system that has the following features. First, it must contain elements with spherical surfaces only. Second, it must contain elements that are chosen from a GRIN catalog. Third, it must require no connectors and the rays should not focus within any elements (since such focusing of a high-powered laser could destroy the element). Finally, of course, it must shape the beam into a uniform profile over the Output Surface. In presenting the problem to the GA, no assumptions were made as to the number of elements required to solve the problem. The GA was allowed to function as a human optical designer would--trying two elements in this system, four in another, and evaluating the merit function in each case to see what improvements, if any, resulted. This problem demonstrates more than any other presented here how the

GA is an example of machine learning. Essentially, information about past successes is preserved in the best individual of each iteration, and this information persists from generation from generation. Progress is made, often in small steps, but sometimes in great leaps (see Fig. 17). The final result is a system not conceived by the human designer, but a system that, nevertheless, functions as required.

Future work on this method lies in two areas. First, solution time must be decreased, which now is on the scale of several hours for the problems considered. This can be accomplished by a number of methods. Because of the nature of the GA process, parallel versions of the code should be relatively easy to implement. This is discussed in detail in Parallelization of the Genetic Algorithm above. Another possibility is to explore implementation of new heuristic models in the GA driver, such as the Tabu search.[35] The GA driver could easily be scaled to a massively-parallel supercomputers, but a raytracing package other than CODE V would have to be used to evaluate the merit function, since there is currently no version of CODE V that runs on supercomputers. A good possibility is KDP[55]. The source code for KDP is available from its developer for a reasonable price.

The second area of work involves applying the GA method to areas other than geometrical optics. These might include solving problems where the geometric approximation fails, such as systems that involve diffractive or holographic elements. To this end, packages other than CODE V would have to be used to evaluate the merit function, since the geometrical irradiance calculation method developed here neglects interference effects. One such package is GLAD, a laser diffraction modeling package, which is developed by Applied Optics Research, 3023 Donee Diego Drive, Escondido, California. If such a problem were pursued, a new interface would have to be written to interface the laser modeling package with the GA driver.

Any of these are worthwhile projects, since this work demonstrates that the GA method can solve many problems in optical design and theoretical optics that are intractable to other methods. Furthermore, the GA method has great flexibility and its range of applicability is broad. After the merit function is properly defined and an efficient method of evaluating the merit function is found, adapting the basic GA kernel to solve the problem at hand follows with ease. In order to solve the problems in this work, it was necessary to develop an efficient, accurate means of calculating irradiance profiles over a variety of surfaces. Building this foundation consumed a large portion of development time for the GA method, but its importance cannot be overstated. The theoretical underpinnings of this irradiance calculation algorithm is presented in Computational Methods for Irradiance Calculations via Ray-trace Methods. In any case, this optimization method provides a means to produce solutions unfettered by convention or human influence.

APPENDIX A: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A GRADIENT-INDEX SHAPER

This appendix illustrates how CODE V can be used to evaluate the merit function for the Gradient-Index system. The code below is written in CODE V macro language. Not shown in this appendix is the GA driver, which is written in Fortran 77. See Ref. 47 for more information on the GA driver. To access the GA driver in CODE V, the GA code was compiled into a library format as specified in the CODE V documentation. See 'User-defined functions' in the CODE V manuals for more information. At line 2, the parameters are defined that provide a communications nexus between CODE V and the GA driver. The GA is invoked at line 20, where the parameters defined in line 2 are passed to the GA. The GA interprets the values stored in the '^fun(200)' array, and uses those in its calculations. In turn, the GA populates the '^par(200,9)' array, which defines a new generation to be evaluated in CODE V. The for..end for loop starting at line 27 and ending at line 111 contains the code that traces the 200 rays for each of the 10 individuals per generation and, most importantly, evaluates the merit function for each. The raytrace is done in lines 67-79. The irradiance profile is calculated in lines 83-92. The precise form of the merit function can be found in lines 95-105. The first term, '1/^merit,' is a value proportional to the flatness (uniformity) of the output profile. '^nsum' is a value related to the angle each ray traced through the system make with the optical ($Z-$) axis. As each of these angles approaches 0, '^nsum' approaches 1. If '^nsum' is one, then it makes no contribution to the merit function. The last term, 'expf(-0.01*((8 - ^r2(200))**2))' is less than one if the radial height of the marginal ray ('^r2(200)') is some value other than eight, penalizing the system. Also note how the Gradient glass type is chosen in lines 47-61. This is a unique feature of this particular Gradient-Index application. The

command 'prv' at line 48 instructs CODE V to begin a private (user-defined) glass type. 'pwl 589' at line 49 indicates that the private glass is valid for wavelength λ=589nm. 'UDG C1' and 'UDG C2' are the two discrete parameters that select the GRIN element from the several available in the CODE V gradient-index catalog. The values are populated from the variables ^UDGC11 and ^UDGC12, which are set by the GA, in lines 52 and 53. This same process is repeated to define a private glass type for the second shaping element in lines 54-58. Finally, the individual glass type for the shaping elements are set to the two newly defined private glasses in lines 60 and 61.

GA macro: ga.seq.1 (CODE V macro language)

```
1   lcl num ^merit ^meritv(10)
2   lcl num ^par(200,9) ^fun(200) ^iv
3   lcl num ^r1(0..200) ^r2(0..200) ^n(0..200) ^srn(0..200)
4   lcl num ^sum ^avg ^eprinc ^raymiss ^nsum
5   lcl num ^i2p ^i2s(0..200)
6   lcl num ^UDGC11 ^UDGC12 ^UDGC21 ^UDGC22
7
8   out n
9   ver all n
10  exc n
11
12  ^eprinc == (epd)/400.
13
14  for ^iv -1 1000000
15
16  if (^iv=0)
17          ^iv == 2
18  end if
19
20  usr ^par ^fun ^iv
21  !out y
22  !wri (tim)
23  !wri "__________"
24  !out n
25
26
27  for ^c 1 10
28
29  thi s2 ^par(^c,1)
30  thi s4 ^par(^c,2)
31  thi s3 ^par(^c,3)
32  rdy s3 ^par(^c,4)
33  rdy s5 ^par(^c,5)
34  ^UDGC11 == roundf(^par(^c,6))
35  if (^par(^c,7) > 0)
36          ^UDGC12 == 1
37  els
38          ^UDGC12 == -1
39  end if
40  ^UDGC21 == roundf(^par(^c,8))
41  if (^par(^c,9) > 0)
42          ^UDGC22 == 1
43  els
44          ^UDGC22 == -1
45  end if
```

```
46
47    del prv all
48    prv
49    pwl 589
50    'n1' LPT GRADIUM 1.7
51    UDG
52    UDG C1 (^UDGC11)
53    UDG C2 (^UDGC12)
54    pwl 589
55    'n2' LPT GRADIUM 1.7
56    UDG
57    UDG C1 (^UDGC21)
58    UDG C2 (^UDGC22)
59    end
60    gla s2 'n1'
61    gla s4 'n2'
62
63    ^merit == 0
64    ^sum == 0
65    ^nsum == 1
66
67    for ^i 0 200
68           ^r1(^i) == ^i*^eprinc
69           ^j == raysin(0,0,0.,^r1(^i),0.,0.)
70           if (^j <> 0)
71                  ^raymiss == 1
72           els
73                  ^r2(^i) == (y s6)
74                  ^r2(^i) == absf(^r2(^i))
75                  ^n(^i) == (n s6)
76                  ^nsum == ^nsum * ^n(^i)**6
77                  ^srn(^i) == (srn s6)
78           end if
79    end for
80
81    if (^raymiss = 0)
82           ^a == -logf((PUI))/((PUX)*(EPD)/2.)**2
83           for ^i 1 200
84                  ^j == ^i-1
85                  ^k == ^i
86                  ^i2p == (^r1(^i)*(^r1(^k)-^r1(^j))*expf(-^a*^r1(^i)**2)*^n(^i))&
87                         /(^r2(^i)*(^r2(^k)-^r2(^j)))
88                  ^i2s(^i) ==
89    (^i2p*^srn(^i))/((^n(^i)*^srn(^i))+(sinf(acosf(^n(^i)))*sinf(acosf(^srn(^i)))))
90           !      ^i2s(^i) == ^i2p*absf(^srn(^i))
91                  ^sum == ^sum + ^i2s(^i)
92           end for
93           ^avg == ^sum/199
94
95           for ^i 1 200
96                  ^merit == ^merit + (^i2s(^i) - ^avg)**2
97           end for
98
99           ^merit == sqrtf(^merit/199)
100
101          ^meritv(^c) == 1/^merit * expf(-1*((1 - ^nsum)**2)) * expf(-0.01*((8 -
102   ^r2(200))**2))
103
104   els
105          ^raymiss == 0
106          ^meritv(^c) == 0
107   end if
108
109   ^fun(^c) == ^meritv(^c)
110
111
112   end for
113
114   end for
115
116   ver y
```

APPENDIX B: CODE SAMPLES FROM DESIGN AND ANALYSIS OF A FREE-FORM GA-DESIGNED GRIN SHAPER

This appendix illustrates how CODE V is used to evaluate the merit function for the Free-Form GA-Designed GRIN. The code below is written in the CODE V macro language as in Appendix A. The functionality is similar to that presented in Appendix A except for several interesting differences. First, the number of elements present in the system is set at line 34 in the variable '^iEle'. Then, ^iEle is checked and so that the appropriate number of surfaces can be nullified where ^iEle<4. For example, if ^iEle=1, the appropriate parameters defining element 1 are set in lines 37-59, while the surfaces defining the remaining three elements are nullified in lines 60-72.

This sequence also uses a new CODE V raytracing function, 'raytra', which appears at line 277. The raytra function offers significantly better performance than the 'raysin' function used at line 66 in Appendix A. Furthermore, special check at lines 289-294 insures that rays do not cross within the system, which is an undesired feature for this problem. As an unexpected consequence, this check offers marginally increased performance since systems with crossing rays are immediately given a merit function value of zero. In the code presented in Appendix A, the merit function for systems with crossing rays would be calculated normally, which led to the focused beam in the connector for the solution presented in Design and Analysis of a Gradient-Index Shaper. The merit function is presented at lines 320-321. This merit function is essentially identical to the one in Appendix A, except for the additional term 'expf(-0.1*((4 - ^r2(1))**2))'. This term was added with the hopes that rays would be pushed from the center of the output irradiance profile more quickly, in order to arrive at a solution more efficiently.

GA macro: ga.seq.7 (CODE V macro language)

```
1    lcl num ^merit ^meritv(10)
2    lcl num ^par(200,26) ^fun(200) ^iv
3    lcl num ^r1(0..200) ^r2(0..200) ^n(0..200) ^srn(0..200)
```

```
lcl num ^sum ^avg ^eprinc ^raymiss ^nsum
lcl num ^i2p ^i2s(0..200)
lcl num ^UDGC11 ^UDGC12 ^UDGC21 ^UDGC22
lcl num ^UDGC211 ^UDGC212 ^UDGC221 ^UDGC222
lcl num ^UDGC311 ^UDGC312 ^UDGC321 ^UDGC322
lcl num ^UDGC411 ^UDGC412 ^UDGC421 ^UDGC422
lcl num ^iEle
lcl num ^input(4) ^output(8)

out n
ver all n
exc n

^eprinc == (epd)/400.

for ^iv -1 1000000

if (^iv=0)
        ^iv == 2
end if

usr ^par ^fun ^iv
!out y
!wri (tim)
!wri "__________"
!out n


for ^c 1 10

^iEle == roundf(^par(^c,1))

if (^iEle = 1)
        rdy s2 ^par(^c,2)
        rdy s3 ^par(^c,3)
        thi s2 ^par(^c,4)
        thi s3 ^par(^c,5)
        ^UDGC11 == roundf(^par(^c,6))
        if (^par(^c,7) > 0)
                ^UDGC12 == 1
        els
                ^UDGC12 == -1
        end if
        gla s2
        gla s4
        gla s6
        gla s8
        del prv all
        prv
        pwl 589
        'n1' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC11)
        UDG C2 (^UDGC12)
        end
        gla s2 'n1'
        rdy s4 9e99
        rdy s5 9e99
        thi s4 0
        thi s5 0
        rdy s6 9e99
        rdy s7 9e99
        thi s6 0
        thi s7 0
        rdy s8 9e99
        rdy s9 9e99
        thi s8 0
        thi s9 0
        thi s10 ^par(^c,26)
els if (^iEle = 2)
        rdy s2 ^par(^c,2)
```

```
        rdy s3 ^par(^c,3)
        thi s2 ^par(^c,4)
        thi s3 ^par(^c,5)
        ^UDGC11 == roundf(^par(^c,6))
        if (^par(^c,7) > 0)
                ^UDGC12 == 1
        els
                ^UDGC12 == -1
        end if
        ^UDGC211 == roundf(^par(^c,12))
        if (^par(^c,13) > 0)
                ^UDGC212 == 1
        els
                ^UDGC212 == -1
        end if
        gla s2
        gla s4
        gla s6
        gla s8
        del prv all
        prv
        pwl 589
        'n1' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC11)
        UDG C2 (^UDGC12)
        pwl 589
        'n2' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC211)
        UDG C2 (^UDGC212)
        end
        gla s2 'n1'
        gla s4 'n2'
        rdy s4 ^par(^c,8)
        rdy s5 ^par(^c,9)
        thi s4 ^par(^c,10)
        thi s5 ^par(^c,11)
        rdy s6 9e99
        rdy s7 9e99
        thi s6 0
        thi s7 0
        gla s6
        rdy s8 9e99
        rdy s9 9e99
        thi s8 0
        thi s9 0
        gla s8
        thi s10 ^par(^c,26)
els if (^iEle = 3)
        rdy s2 ^par(^c,2)
        rdy s3 ^par(^c,3)
        thi s2 ^par(^c,4)
        thi s3 ^par(^c,5)
        ^UDGC11 == roundf(^par(^c,6))
        if (^par(^c,7) > 0)
                ^UDGC12 == 1
        els
                ^UDGC12 == -1
        end if
        ^UDGC211 == roundf(^par(^c,12))
        if (^par(^c,13) > 0)
                ^UDGC212 == 1
        els
                ^UDGC212 == -1
        end if
        ^UDGC311 == roundf(^par(^c,18))
        if (^par(^c,19) > 0)
                ^UDGC312 == 1
        els
                ^UDGC312 == -1
```

```
146         end if
147
148         gla s2
149         gla s4
150         gla s6
151         gla s8
152         del prv all
153         prv
154         pwl 589
155         'n1' LPT GRADIUM 1.7
156         UDG
157         UDG C1 (^UDGC11)
158         UDG C2 (^UDGC12)
159         'n2' LPT GRADIUM 1.7
160         UDG
161         UDG C1 (^UDGC211)
162         UDG C2 (^UDGC212)
163         pwl 589
164         'n3' LPT GRADIUM 1.7
165         UDG
166         UDG C1 (^UDGC311)
167         UDG C2 (^UDGC312)
168         end
169         gla s2 'n1'
170         rdy s4 ^par(^c,8)
171         rdy s5 ^par(^c,9)
172         thi s4 ^par(^c,10)
173         thi s5 ^par(^c,11)
174         gla s4 'n2'
175         rdy s6 ^par(^c,14)
176         rdy s7 ^par(^c,15)
177         thi s6 ^par(^c,16)
178         thi s7 ^par(^c,17)
179         gla s6 'n3'
180         rdy s8 9e99
181         rdy s9 9e99
182         thi s8 0
183         thi s9 0
184         gla s8
185         thi s10 ^par(^c,26)
186   els
187         rdy s2 ^par(^c,2)
188         rdy s3 ^par(^c,3)
189         thi s2 ^par(^c,4)
190         thi s3 ^par(^c,5)
191         ^UDGC11 == roundf(^par(^c,6))
192         if (^par(^c,7) > 0)
193                 ^UDGC12 == 1
194         els
195                 ^UDGC12 == -1
196         end if
197         ^UDGC211 == roundf(^par(^c,12))
198         if (^par(^c,13) > 0)
199                 ^UDGC212 == 1
200         els
201                 ^UDGC212 == -1
202         end if
203         ^UDGC311 == roundf(^par(^c,18))
204         if (^par(^c,19) > 0)
205                 ^UDGC312 == 1
206         els
207                 ^UDGC312 == -1
208         end if
209         ^UDGC141 == roundf(^par(^c,24))
210         if (^par(^c,25) > 0)
211                 ^UDGC412 == 1
212         els
213                 ^UDGC412 == -1
214         end if
215
216
```

```
        gla s2
        gla s4
        gla s6
        gla s8
        del prv all
        prv
        pwl 589
        'n1' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC11)
        UDG C2 (^UDGC12)
        pwl 589
        'n2' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC211)
        UDG C2 (^UDGC212)
        pwl 589
        pwl 589
        'n3' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC311)
        UDG C2 (^UDGC312)
        pwl 589
        'n4' LPT GRADIUM 1.7
        UDG
        UDG C1 (^UDGC411)
        UDG C2 (^UDGC412)
        end
        gla s2 'n1'
        rdy s4 ^par(^c,8)
        rdy s5 ^par(^c,9)
        thi s4 ^par(^c,10)
        thi s5 ^par(^c,11)
        prv
        pwl 589
        end
        gla s4 'n2'
        rdy s6 ^par(^c,14)
        rdy s7 ^par(^c,15)
        thi s6 ^par(^c,16)
        thi s7 ^par(^c,17)
        gla s6 'n3'
        rdy s8 ^par(^c,20)
        rdy s9 ^par(^c,21)
        thi s8 ^par(^c,22)
        thi s9 ^par(^c,23)
        gla s8 'n4'
        thi s10 ^par(^c,26)
end if


^merit == 0
^sum == 0
^nsum == 1

!for ^i 0 200
^i == 200
unt
        ^r1(^i) == ^i*^eprinc
        ^input(1)==0; ^input(2)==^r1(^i); ^input(3)==0; ^input(4)==0
        ^j == raytra(0,0,0,^input,^output)
        if (^j <> 0)
                ^raymiss == 1
                ^i == 0
        els
                !^r2(^i) == (y s11)
                ^r2(^i) == ^output(2)
                ^r2(^i) == absf(^r2(^i))
                !^n(^i) == (n s11)
                ^n(^i) == ^output(6)
                ^nsum == ^nsum * ^n(^i)**6
```

```
                ^srn(^i) == (srn s11)
                if ^i < 200
                        if ^r2(^i) > ^r2(^i+1)
                                ^raymiss == 1
                                ^i == 0
                        end if
                end if
        end if
!end for
        ^i == ^i-1
end unt (^i < 0)

if (^raymiss = 0)
        ^a == -logf((PUI))/((PUX)*(EPD)/2.)**2
        for ^i 1 200
                ^j == ^i-1
                ^k == ^i
                ^i2p == (^r1(^i)*(^r1(^k)-^r1(^j))*expf(-^a*^r1(^i)**2)*^n(^i))&
                        /(^r2(^i)*(^r2(^k)-^r2(^j)))
                ^i2s(^i) ==
(^i2p*^srn(^i))/((^n(^i)*^srn(^i))+(sinf(acosf(^n(^i)))*sinf(acosf(^srn(^i)))))
        !       ^i2s(^i) == ^i2p*absf(^srn(^i))
                ^sum == ^sum + ^i2s(^i)
        end for
        ^avg == ^sum/199

        for ^i 1 200
                ^merit == ^merit + (^i2s(^i) - ^avg)**2
        end for

        ^merit == sqrtf(^merit/199)

        ^meritv(^c) == 1/^merit * expf(-1*((1 - ^nsum)**2)) * expf(-0.01*((8 -
^r2(200))**2))* expf(-0.1*((4 - ^r2(1))**2))

els
        ^raymiss == 0
        ^meritv(^c) == 0
end if

^fun(^c) == ^meritv(^c)


end for

end for

ver y
```

APPENDIX C: CODE SAMPLES FROM THE FORTRAN GENETIC ALGORITHM DRIVER

This appendix illustrates how many of the GA techniques actually are coded. As mentioned in Appendix A, the GA driver is written in Fortran 77 and compiled into an object as specified in the CODE V manual. The subroutine in this object called 'USERSUB' (as required by CODE V) is than instatiated when the appropriate command is issued in a CODE V macro sequence (see line 20 in Appendix A and line 25 in Appendix B). The variables used to pass information to and from the GA driver and CODE V must be defined appropriately in both the GA driver and in the CODE V macro sequence. In the applications presented in this work, three such variables were used: 'parentv', an array containing the values for all parameters to be optimized for each system in a generation (or step); 'funcvalv', an array containing the values of the merit functions for each system in a generation; and 'iv', an integer indicating the generation number. These variables can be seen in lines 1-2 below.

Fundamental to the GA process is the ability to encode real numbers into strings, which are the genetic material for a system. The code used to encode and decode the parameters into and from strings is shown in lines 5-71. Also, two important operators, 'crossover' and 'mutate' are coded in lines 72-194. Finally, the Micro-GA code, which checks for 'stagnancy' among adjacent populations, is shown in lines 195-246. The Micro-GA function allows for small populations sizes (usually 10 members), which is important for these applications since evaluation of the merit function is so time-consuming. It should be noted that the code presented below is only a small part of the entire Fortran GA driver, since the entire code set is quite large. Contact the author for a complete copy of the GA driver.

GA driver: ga164.f (Fortran 77)

```
subroutine USERSUB ( parentv, funcvalv, iv)
      REAL*8 parentv(indmax,nparmax),funcvalv(indmax),iv
      INTEGER*4 res

c#########################################################################
      subroutine decode(i,array,iarray)
c
c  This routine decodes a binary string to a real number.
c
      implicit double precision (a-h,o-z)
      save
c
      include 'params.f'
      common / ga2   / nparam,nchrome
      common / ga5   / g0,g1,ig2
      dimension array(indmax,nparmax),iarray(indmax,nchrmax)
      dimension g0(nparmax),g1(nparmax),ig2(nparmax)
c
      l=1
      do 10 k=1,nparam
         iparam=0
         m=l
         do 20 j=m,m+ig2(k)-1
            l=l+1
            iparam=iparam+iarray(i,j)*(2**(m+ig2(k)-1-j))
 20      continue
         array(i,k)=g0(k)+g1(k)*dble(iparam)
 10   continue
c
      return
      end
c
c#########################################################################
      subroutine code(j,k,array,iarray)
c
c  This routine codes a parameter into a binary string.
c
      implicit double precision (a-h,o-z)
      save
c
      include 'params.f'
      common / ga2   / nparam,nchrome
      common / ga5   / g0,g1,ig2
      dimension array(indmax,nparmax),iarray(indmax,nchrmax)
      dimension g0(nparmax),g1(nparmax),ig2(nparmax)
c
c  First, establish the beginning location of the parameter string of
c  interest.
      istart=1
      do 10 i=1,k-1
         istart=istart+ig2(i)
 10   continue
c
c  Find the equivalent coded parameter value, and back out the binary
c  string by factors of two.
      m=ig2(k)-1
      if (g1(k).eq.0.0) return
      iparam=nint((array(j,k)-g0(k))/g1(k))
      do 20 i=istart,istart+ig2(k)-1
         iarray(j,i)=0
```

```
         if ((iparam+1).gt.(2**m)) then
            iarray(j,i)=1
            iparam=iparam-2**m
         endif
         m=m-1
 20   continue
c     write(3,*)array(j,k),iparam,(iarray(j,i),i=istart,istart+ig2(k)-1)
c
      return
      end

c*********************************************************************
      subroutine crosovr(ncross,j,mate1,mate2)
c
c  Subroutine for crossover between the randomly selected pair.
      implicit double precision (a-h,o-z)
      save
c
      include 'params.f'
      dimension parent(indmax,nparmax),child(indmax,nparmax)
      dimension iparent(indmax,nchrmax),ichild(indmax,nchrmax)
c
      common / ga2   / nparam,nchrome
      common / ga3   / parent,iparent
      common / ga7   / child,ichild
      common /inputga/ pcross,pmutate,pcreep,maxgen,idum,irestrt,
     +                 itourny,ielite,icreep,iunifrm,iniche,
     +                 iskip,iend,nchild,microga,kountmx
c
      if (iunifrm.eq.0) then
c  Single-point crossover at a random chromosome point.
         call ran3(1,rand)
         if(rand.gt.pcross) goto 69
         ncross=ncross+1
         call ran3(1,rand)
         icross=2+dint(dble(nchrome-1)*rand)
         do 50 n=icross,nchrome
            ichild(j,n)=iparent(mate2,n)
            if(nchild.eq.2) ichild(j+1,n)=iparent(mate1,n)
 50      continue
      else
c  Perform uniform crossover between the randomly selected pair.
         do 60 n=1,nchrome
            call ran3(1,rand)
            if(rand.le.pcross) then
               ncross=ncross+1
               ichild(j,n)=iparent(mate2,n)
               if(nchild.eq.2) ichild(j+1,n)=iparent(mate1,n)
            endif
 60      continue
      endif
 69   continue
c
      return
      end
c
c*********************************************************************
      subroutine mutate
c
      implicit double precision (a-h,o-z)
      save
c
      include 'params.f'
```

```
      dimension nposibl(nparmax)
      dimension child(indmax,nparmax),ichild(indmax,nchrmax)
      dimension g0(nparmax),g1(nparmax),ig2(nparmax)
      dimension parmax(nparmax),parmin(nparmax),pardel(nparmax)
c
      common / ga1   / npopsiz,nowrite
      common / ga2   / nparam,nchrome
      common / ga5   / g0,g1,ig2
      common / ga6   / parmax,parmin,pardel,nposibl
      common / ga7   / child,ichild
      common /inputga/ pcross,pmutate,pcreep,maxgen,idum,irestrt,
     +                 itourny,ielite,icreep,iunifrm,iniche,
     +                 iskip,iend,nchild,microga,kountmx
c
c  This subroutine performs mutations on the children generation.
c  Perform random jump mutation if a random number is less than pmutate.
c  Perform random creep mutation if a different random number is less
c  than pcreep.
      nmutate=0
      ncreep=0
      do 70 j=1,npopsiz
         do 75 k=1,nchrome
c  Jump mutation
            call ran3(1,rand)
            if (rand.le.pmutate) then
               nmutate=nmutate+1
               if(ichild(j,k).eq.0) then
                  ichild(j,k)=1
               else
                  ichild(j,k)=0
               endif
               if (nowrite.eq.0) write(6,1300) j,k
               if (nowrite.eq.0) write(24,1300) j,k
            endif
 75      continue
c  Creep mutation (one discrete position away).
         if (icreep.ne.0) then
            do 76 k=1,nparam
               call ran3(1,rand)
               if(rand.le.pcreep) then
                  call decode(j,child,ichild)
                  ncreep=ncreep+1
                  creep=1.0
                  call ran3(1,rand)
                  if (rand.lt.0.5) creep=-1.0
                  child(j,k)=child(j,k)+g1(k)*creep
                  if (child(j,k).gt.parmax(k)) then
                     child(j,k)=parmax(k)-1.0*g1(k)
                  elseif (child(j,k).lt.parmin(k)) then
                     child(j,k)=parmin(k)+1.0*g1(k)
                  endif
                  call code(j,k,child,ichild)
                  if (nowrite.eq.0) write(6,1350) j,k
                  if (nowrite.eq.0) write(24,1350) j,k
               endif
 76         continue
         endif
 70   continue
      write(6,1250) nmutate,ncreep
      write(24,1250) nmutate,ncreep
c
 1250 format(/'  Number of Jump Mutations  =',i5/
     +        '  Number of Creep Mutations =',i5)
```

```
 1300 format('*** Jump mutation performed on individual  ',i4,
     +       ', chromosome ',i3,' ***')
 1350 format('*** Creep mutation performed on individual ',i4,
     +       ', parameter  ',i3,' ***')
c
      return
      end
c
c#######################################################################
      subroutine gamicro(i,npossum,ig2sum,ibest)
c
c  Micro-GA implementation subroutine
c
      implicit double precision (a-h,o-z)
      save
c
      include 'params.f'
      dimension parent(indmax,nparmax),iparent(indmax,nchrmax)
      dimension ibest(nchrmax)
c
      common / ga1   / npopsiz,nowrite
      common / ga2   / nparam,nchrome
      common / ga3   / parent,iparent
c
c  First, check for convergence of micro population.
c  If converged, start a new generation with best individual and fill
c  the remainder of the population with new randomly generated parents.
c
c  Count number of different bits from best member in micro-population
      icount=0
      do 81 j=1,npopsiz
         do 82 n=1,nchrome
            if(iparent(j,n).ne.ibest(n)) icount=icount+1
 82      continue
 81   continue
c
c  If icount less than 5% of number of bits, then consider population
c  to be converged.  Restart with best individual and random others.
      diffrac=dble(icount)/dble((npopsiz-1)*nchrome)
      if (diffrac.lt.0.05) then
      do 87 n=1,nchrome
         iparent(1,n)=ibest(n)
 87   continue
      do 88 j=2,npopsiz
         do 89 n=1,nchrome
            call ran3(1,rand)
            iparent(j,n)=1
            if(rand.lt.0.5) iparent(j,n)=0
 89      continue
 88   continue
      if (npossum.lt.ig2sum) call possibl(parent,iparent)
      write(6,1375) i
      write(24,1375) i
      endif
c
 1375 format(//'%%%%%%%  Restart micro-population at generation',
     +       i5,'  %%%%%%%')
c
      return
      end
```

LIST OF REFERENCES

1. Pablo Neruda, Twenty Love Poems and a Song of Despair (Chronicle Books, San Francisco, 1993) p. 17.

2. Or, using more conventional optics terminology, a multi-dimensional 'merit function' which quantifies the worth of an optical system in terms of the various parameters which define the system.

3. M. Gell-Mann, *The Quark and the Jaguar* (W.H. Freeman, New York, 1994) Chap. 20, p. 312.

4 . D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, MA, 1989) Chap 1, p. 1-6.

5 . B. P. Buckles and F. E. Petry, *Genetic Algorithms* (IEE Computer Society Press, Los Alamitos, CA, 1992) p. 1.

6. D. E. Goldberg, p. 7.

7. J. Bengtsson, "Kinoform-only Gaussian-to-rectangle beam shaper for a semiconductor laser," Appl. Opt. 35, 3807-3814 (1996).

8. C. Han, Y. Ishii, and K. Murata, "Reshaping collimated laser beams with Gaussian profile to uniform profiles," Appl. Opt. 22, 3644-3647 (1983).

9. X. Tan, B. Gu, G. Yang, and B. Dong, "Diffractive phase elements for beam shaping: a new design method," Appl. Opt. 34, 1314-1320 (1995).

10. F. M. Dickey and S. C. Holswade, "Gaussian laser beam profile shaping," Opt. Eng. 35 3285-3295 (1996).

11. P. W. Rhodes and D. L. Shealy, "Refractive optical systems for irradiance redistribution of collimated radiation: their design and analysis," Appl. Opt. 19, 3545-3553 (1980).

12. D. Shafer, "Gaussian to flat-top intensity distributing lens," Opt. Laser Technol. 14, 159-160 (1982).

13. P. H. Malyak, "Two-mirror unobscured optical system for reshaping the irradiance redistribution of a laser beam," Appl. Opt. 31, 4377-4383 (1992).

14. N. C. Evans and D. L. Shealy, "Design And Optimization Of An Irradiance Profile-Shaping System With A Genetic Algorithm Method," Appl. Opt. 37, 5216-5221 (1998).

15. M. Kuittinen, P. Vahimaa, M. Honkanen, and J. Turunen, "Beam shaping in the nonparaxial domain of diffractive optics," Appl. Opt. 36, 2034-2041 (1997).

16. W. Jiang, D. L. Shealy, J. C. Martin, "Design and testing of a refractive reshaping system," in *Current Developments in Optical Design and Optical Engineering III*, Robert E. Fischer; Warren J. Smith, eds., Proc. SPIE 2000, 64-75 (1993).

17. C. Wang and D. L. Shealy, "Design of gradient-index lens systems for laser beam reshaping," Appl. Opt. **32**, 4763-4769 (1993).

18. E. Betensky, "Postmodern lens design," Opt. Eng. **32**, 1750-1756 (1993).

19. K. Nemoto, T. Nayuki, T. Fujii, N. Goto, and Y. Kanai, "Optimum control of the laser beam intensity profile with a deformable mirror," Appl. Opt. **36**. 7689-7695 (1997).

20. J. H. McDermit and T. E. Horton, "Reflective Optics for Obtaining Prescribed Irradiative Distributions from Collimated Sources," Appl. Opt. 13, 1444-1450 (1974).

21. V. Oliker, L. Prussner, D. L. Shealy and S. Mirov, "Optical Design of a two-mirror symmetrical reshaping system and its application in superbroadband color center laser," in *Current Developments in Optical Design and Optical Engineering IV*, Robert E. Fischer; Warren J. Smith, eds., Proc. SPIE 2263, 10-18 (1994).

22. T. Dresel, M. Beyerlein and J. Schwider, "Design of computer generated beam-shaping holograms by iterative finite-element mesh adaptation," Appl. Opt. 35, 6865-6874 (1996).

23. P. Mouroulis and J. Macdonald, Geometrical Optics and Optical Design, Oxford University Press, New York (1997).

24. M. Born and E. Wolf, Principles of Optics, Fifth Edition, Pergamon Press, New York (1975).

25. W.T. Welford, Aberrations of the Symmetrical Optical System, Academic Press, New York (1974).

26. G.A. Deschamps, ``Ray Techniques in Electromagnetics," Proc. IEEE 60.9, 1022-1035 (1972).

27. D. Shealy, “Geometrical Methods,” in *Laser Beam Shaping Theory and Techniques*, Fred M. Dickey; Scott C. Holswade, eds. (in press, Marcel Decker, NY, 2000).

28. S. Solimeno, B. Crosignani, and P. DiPorto, *Guiding, Diffraction, and Confinement of Optical Radiation* (Academic Press, Orlando, 1984), p. 49.

29. A.K. Ghatak and K. Thyagarajan, *Contemporary Optics* (Plenum Press, New York, 1980), p. 24.

30. F.A. Jenkins and H.E. White, *Fundamentals of Optics* (McGraw-Hill Book Company, Inc., New York, 1957), p. 481.

31. Born and Wolf, p. 115.

32. W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, Numerical Recipes in Fortran 77: The Art of Scientific Computing (Cambridge U.P., Cambridge, U.K., 1992) Chap. 10, p. 387-388.

33. Dennis, J.E. and Schnabel, R.B., Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Prentice-Hall, Englewood Cliffs, NJ, 1983).

34. Brent, R.P., Algorithms for Minimization without Derivatives (Prentice-Hall, Englewood Cliffs, NJ, 1983).

35. L. B. Morales, ``Scheduling a bridge club by tabu search," Mathematics Magazine **70**, 287-290 (1997).

36. W. H. Press, p. 436-437.

37. G. B. Dantzig, Linear Programming and Extensions (Princeton U.P., Princeton, N. J., 1963.

38. T. Etzion and P.R.J. Ostergand, “Greedy and Heuristic Algorithms for Codes and Colorings,” IEEE Transactions on Information Theory 44, 382-388 (1988).

39. R.H.J.M. Otten and L.P.P.P. van Ginneken, *The Annealing Algorithm* (Kluwer, Boston, 1989).

40. R. R. Brooks, S. S. Iyengar and J. Chen, “Self-Calibration of a Noisy Mulitple-Sensor System with Genetic Algorithms,” in *Self-Calibrated Intelligent Optical Sensors and Systems*, Anbo Wang, ed., Proc. SPIE 2594, 20-38 (1996).

41. H. Kim, Y. Hayashi and K. Nara, “An Algorithm for Thermal Unit Maintenance Scheduling through Combined Use of GA, SA and TS,” IEEE Transactions on Power Systems 12, 329-335 (1197).

42. F. S. Wen and C. S. Chang, "Tabu Search Approach to Alarm Processing in Power Systems," IEEE Proceedings: Generations, Transmission and Distribution 144, 31-38 (1997).

43. H. J. Jensen, *Self-Organized Criticality: Emergent Complex Behavior in Physical and Biological Systems* (Cambridge U. P., New York, 1998) p 34-71.

44. OSLO is a registered trademark of Sinclair Optics, Inc., 6780 Palmyra Road, Fairport, NY, 14450.

45. ZEMAX is a registered trademark of Focus Software, Inc., P.O. Box 18228, Tucson, Arizona, 85731.

46. CODE V is a registered trademark of Optical Research Associates, 3280 E. Foothill Blvd., Pasadena, CA, 91107.

47. The GA code is based on ga164.f, D.L. Carroll's FORTRAN Genetic Algorithm Driver. See http://www.staff.uiuc.edu/~carroll/ga.html to download ga164.f, along with documentation and useful information on its implementation. One may find several sites related to ga164.f by using it as a keyword on popular search engines. Ga164.f is based on a Micro-GA method. See Ref. 49 for more information on Micro-GAs.

48. See the Naval Research Lab's GA archive, http://www.aic.nrl.navy.mil/galist/, for a comprehensive source of GA codes in several different languages and implementations. This site also is the home of a very useful mailing list in which one may find the current state of the art in GA techniques and theory.

49. K. Krishnakumar, "Micro-Genetic Algorithms for Stationary and Non-Stationary Function Optimization," SPIE: Intelligent Control and Adaptive Systems 1196, (1989).

50. D. E. Goldberg, p. 21.

51. P. Kraft, N. R. Harvey, S. Marshall, "Parallel genetic algorithms in the optimization of morphological filters," J. of Electronic Imaging 6, 504-516 (1997).

52. A. B. Djuriši, J. M. Elazar, A. D. Raki, "Simulated-annealing-based genetic algorithm for modeling the optical constants of solids," Appl. Opt. 36, 7097-7103 (1997).

53. C. H. Still, "Portable Parallel Computing via the MPI-2 Message-Passing Standard,"Computers in Physics 8, 533-8 (1994).

54. D. W. Walker, "The Design of a Standard Message-Passing Interface for Distributed Memory Concurrent Computers," Parallel Computing 20, 657-673 (1994).

55. It should be noted that there is at least one optical design package for which the source code is available. KDP can be downloaded in compiled form for no charge from http://www.kdpoptics.com. The full source for KDP is available for a reasonable price. Conceivably, one could use KDP for the ray-tracing aspects of the merit function evaluation, and modify the KDP source code so it can be compiled in a supercomputer environment.

56. K. M. Baker, D. L. Shealy, W. Jiang, "Directional light filters: three-dimensional azo dye formed micro-honeycomb images with optical resins," in *Diffractive and Holographic Optics Technology II*, I. Cindrich; S. H. Lee, eds., Proc. SPIE 2404, 144-158 (1995).

57. Ken Baker of Optimetrix Co. provided the initial specifications and requirements for this system. Optimetrix is located at 13659 Victory Blvd., Van Nuys, CA, 91401. The final has been fabricated by Optimetrix as part of a holographic projection system. See K. M. Baker, "Highly Corrected Submicrometer Grid Patterning on Curved Surfaces," Appl. Opt. **38**, 339-351 (1999).

58. P. Mouroulis and J. Macdonald, *Geometrical Optics and Optical Design* (Oxford Univ. Press, New York, 1997) Chap. 9, p. 248-250.

59. W. Jiang, *Application of a Laser Beam Profile Reshaper to Enhance Performance of Holographic Projection Systems* (Univ. of Ala. at Birmingham Dept. of Physics, Birmingham, 1993) Chap 2, p.30-32.

60. W. H. Press, p. 425-436.

61. For continuous parameters, numbers are real and are in units of mm.

62. 'UDG C1' is a CODE V variable that refers to 'User-defined glass constant 1'. C1 corresponds to a particular GRIN in the CODE V Lightpath catalog. The CODE V Lightpath GRIN glass catalog can be obtained from Optical Research Associates.[46]