JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, December 2005

HiPPO: An Object-Oriented Framework for General-Purpose Design Optimization

Hongbing Fang* and Mark F. Horstemeyer†
Mississippi State University, Mississippi State, MS, 39762

In this article, we present an object-oriented (OO) optimization framework (OF) incorporated with a generic optimizer interface (GOI) to bridge the gap between different optimization problems and existing optimizers. The OF is a general-purpose system integrating different methods of design of experiments, metamodeling, and multi-objective optimization. By using symbolically defined optimization functions, the OF eliminates the need for user programming and can solve the same problem with any optimizer in the OF. The OF is designed with a graphical user interface (GUI), and utilizes the multi-threading mechanism to overcome the interface-locking problem common to GUI applications. Both the OF and GOI are designed with the OO concept to achieve high maintainability and extensibility. This OF has been successfully used in various optimization applications.

I. Introduction

DESIGN optimization has been playing an important role in the fields of aerospace, civil, mechanical, and other engineering disciplines. Over the years, researchers and engineers have developed various optimization procedures (optimizers) that were successfully used in solving a wide range of optimization problems. One special feature of optimization is that this mathematical process involves function evaluations of objectives, constraints, and their gradients (for gradient-based methods). The objective and constraint functions are problem specific and unknown to the optimizers that have already been developed. For this reason, it is not uncommon to see that many optimizers are provided as a procedure rather than a complete system, because recompilations are needed after combining an optimizer with the user-provided functions typically in the format of computer programs. User programming needs to conform to the specifications of the optimizer's application program interface (API). Since different optimizers typically have different APIs, the user-programming task needs to be repeated if a difficult optimizer is to be used to solve the same problem. In this situation, an optimizer is not a functional system until it combines with a user-provided function. Such optimizers have a problem-dependency and are only procedures rather than general-purpose systems.

With the continuous increase in the complexity of engineering problems, it is often very difficult to obtain the explicit formulations of objective and constraint functions. In addition, the amount of time for solving such complex problems is typically huge, even with the aid of supercomputers and parallel processing. For these reasons, the metamodeling-based optimization approach has been widely adopted, and it has been shown to give reasonably accurate solutions for many engineering applications if properly used. In the metamodeling approach, a certain number of designs are pre-chosen using methods from design of experiments (DOE), and the responses at these design points are obtained through experiments or simulations. The approximate functions of the true responses are then constructed using metamodeling methods such as the polynomial regression (PR) method[1] and radial basis functions (RBF).[2] Such an approximate function is called a metamodel that has a predefined format but

Received 25 May 2005; revision received 25 October 2005; accepted for publication 28 October 2005.

* Assistant Research Professor, Center for Advanced Vehicular Systems, P.O. Box 5405, Member AIAA.
† Professor, Department of Mechanical Engineering, P.O. Box ME.

unknown coefficients to be determined by data at the design points. Since all of the functions are in the known format of the metamodeling method, we can combine metamodeling methods with optimization procedures to solve the problem-dependency. Such a combination generates an optimization system that does not require user programming and program recompilation, if all of the optimization functions are created with the metamodeling methods. We have seen good examples of this approach in commercial software programs such as VisualDoc,[3] iSight,[4] OptiStruct,[5] LS-OPT,[6] and PAM-OPT.[7] Such systems also provide API for user-defined functions, because this is needed in many situations. However, when user-defined functions are used through the APIs, these systems lose their generality.

Many engineering software systems achieve problem-independency by using well-defined formats in describing a problem. One good example is the finite element analysis (FEA) programs in which all problems are discretized and represented by nodes, elements, material properties, boundary conditions, and load cases. Changing a problem only requires changing the problem definition, but does not require changing the FEA system, as long as the problem to be solved remains within the scope of the FEA program. If we can define any optimization problems symbolically rather than using coding, and provide a generic interface from which all optimizers can dynamically obtain values of objective and constraint functions, we can eliminate this problem-dependency.

We define a general-purpose optimization system as one that can perform DOE and metamodeling for problems whose explicit functions are either unavailable or very expensive to obtain. Such a system also needs to support problems with user-provided functions, preferably in the same input format as those for the metamodel functions. Finally, such an optimization system needs to support various existing optimizers so that they can all be used on a given optimization problem that has been defined. According to the no-free-lunch-theorems for optimization,[8] there is no single optimization algorithm that is best for all types of problems. Therefore, the third feature is highly desirable for a general-purpose optimization system. To achieve this, we need a generic optimizer interface (GOI) to bridge the unknown optimization problems with the various existing optimizers. With the GOI, all problems are defined symbolically in the GOI's format as if there were a single "generic" optimizer; all optimizers can be used to solve a given problem by obtaining function values through the GOI without knowing the function types. To the best of the authors' knowledge, an optimization system with the aforementioned features has not been found in the literature.

The object-oriented (OO) design concept has been increasingly used in various engineering fields during the past several years. The OO concept provides clear system designs, expedites system implementation, and increases system maintainability and extensibility. In this article, we present an OO optimization framework (OF) named HiPPO that integrates various DOE methods, metamodeling methodologies, and a GOI-based multi-objective optimizer that can interface with different optimization methods. All these functionalities are provided under HiPPO's graphical user interface (GUI). HiPPO was designed with the multi-threading mechanism that not only improves system performance, but also solves the interface-locking problem commonly found in GUI applications. In the remaining portion of this article, we first give an overview of a general-purpose optimization system. We then present the system designs of both the OF and the GOI-based optimizer, followed by details of system implementation. Finally, we illustrate the functionalities of HiPPO with two optimization problems followed by some conclusions.

II. Overview of a General-Purpose Optimization Framework

We define a general-purpose OF as one that does not require user-programming and program recompilation for any given optimization problems, that provides DOE and metamodeling capabilities when the optimization functions do not have explicit forms, and that can solve any given problem using different optimizers without redefining the problem. Figure 1 shows the overall design of HiPPO as an example of such a general-purpose OF.

There are three major components in HiPPO, DOE, metamodeling, and optimization. These components are integrated into the OF, but are loosely connected through text files that can be modified if necessary. Loose coupling between functional modules has the advantage of flexibility and modular independence that allow us to start from any point within the entire process of design optimization. For example, we can start from the optimization module if explicit functions are available; otherwise we can start from metamodeling with available or new design samples to create metamodels.

Another flexibility is that we can create a metamodel using any available metamodeling method for a given set of design points that may or may not be generated by DOE, though using DOE is recommended. In addition, we can

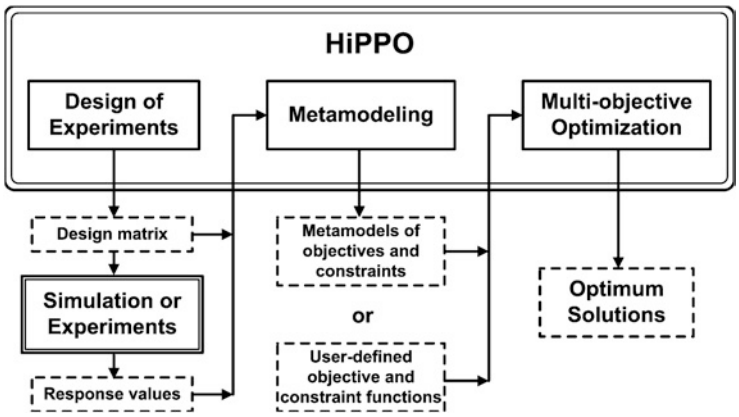

Fig. 1 An example of a general-purpose design optimization framework.

add more design points at a later time to an existing design matrix and create new metamodels. Whether functions are provided by users or from metamodeling, different optimizers and multi-objective formulations can be applied to solve the given problem. The functional module "multi-objective optimization" in Fig. 1 actually represents a GOI-based multi-objective optimizer that integrates multiple optimizers; the GOI-based optimizer is shown in Fig. 2.

All input functions to the GOI-based optimizer are converted into a standard format called internal function representation (IFR) for fast evaluation and manipulation. For multi-objective optimization, we can apply different

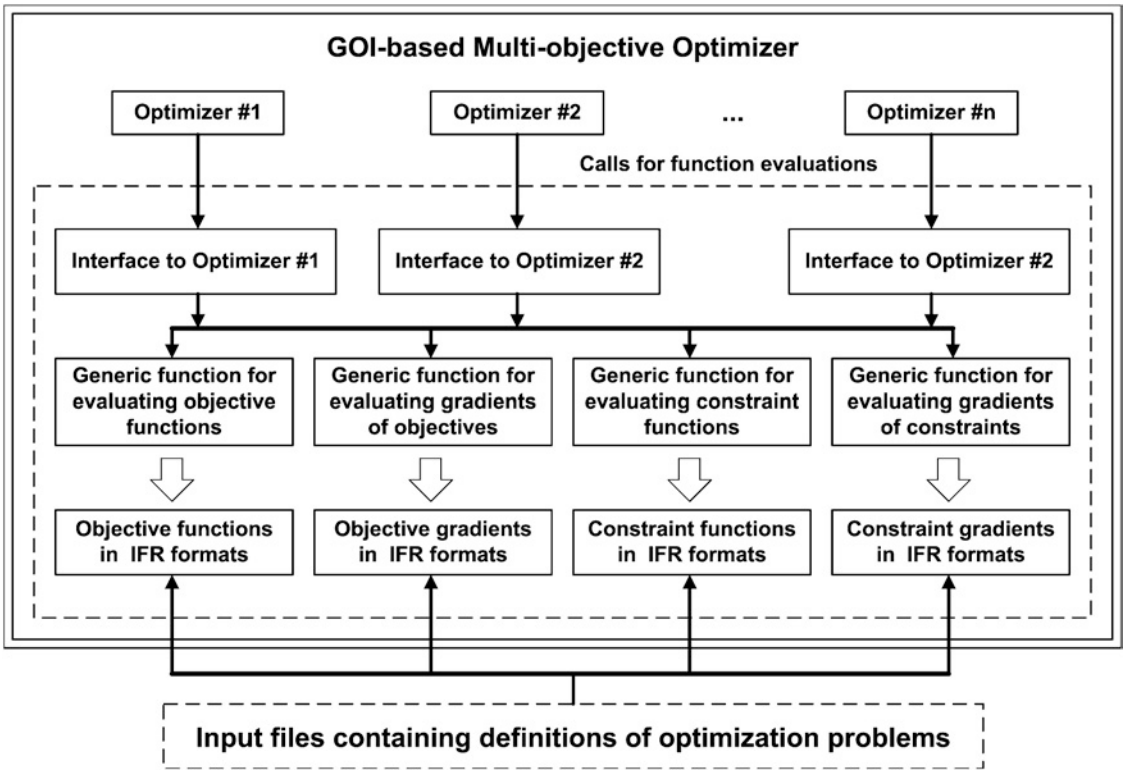

Fig. 2 The GOI-based multi-objective optimizer.

formulations such as weighted sum, global criteria, or nonlinear formulation to the IFRs to form a new objective function if a single-objective optimizer is to be used. In the optimization system of Fig. 2, the GOI provides a special interface to each of the optimizers. During the optimization process, when the selected optimizer requests for values of objectives, constraints, and/or gradients, the generic function evaluation routines in the GOI are called from the special interface and calculate function values using the IFRs. Note that only the gradient-based optimizers need the gradient values.

We only need to define a new special interface if a new optimizer is to be added. The new optimizer only interfaces with the GOI and uses the IFRs; therefore, it can be used to solve any existing and future optimization problems. The mechanism shown in Fig. 2 successfully bridges the gap between various existing optimizers and different optimization problems.

III. System Design

In OO design, we combine a certain type of data with the operations for data manipulation to form an entity that is called an object. This combination is formally referred to as data encapsulation and the prototype for defining such an object is called a class. For example, the data for defining a two-dimensional rectangle can be the coordinates of its bottom-left corner and the lengths of two adjacent sides, and the operations can be draw, move, and erase. Another important feature of OO design is inheritance, which means that a child class can inherit data and/or operations from its parent class in addition to its own.

System design is a high-level overview of the functionalities, organizations, and relationships of all classes independent of the system implementation. In this section, we present design details and the multi-threading mechanism used in HiPPO. Since the "multi-objective optimization" module in HiPPO is an independent subsystem called the GOI-based multi-objective optimizer, we also give its system design in Part C of this section.

A. Class Organization of HiPPO

To enhance the clarity of class organization, we put classes either closely related or with similar functionalities into a collection that is formally called a package. Packages can be organized hierarchically, that is, a package can have sub-packages. Packages are not indispensable to OO design; however, they provide better system clarity and protect class visibility. Figure 3 shows the hierarchical organization of packages and classes in HiPPO.

The top-level package *HiPPO* contains a single class *HippoMain*, which is the program entry and user interface (UI) initializer. Package *HiPPO* has three sub-packages, *HiPPO.UserInterface*, *HiPPO.Utility*, and *HiPPO.TaskPanels*. Package *HiPPO.UserInterface* contains classes for defining GUI components such as menus, toolbars, action handlers, etc. The *HiPPO.Utility* package contains classes that provide matrix manipulation, polynomial functions, regression methods, radial basis functions, and file input/output utilities. Package *HiPPO.TaskPanels* contains the three major components in HiPPO as shown in Fig. 1; they are given in three packages, *HiPPO.TaskPanels.DOE*, *HiPPO.TaskPanels.MMD*, and *HiPPO.TaskPanels.OPT* for performing DOE, metamodeling, and optimization, respectively. Package *HiPPO.TaskPanels.WRT* provides a text editor and is not a key component to HiPPO.

The two classes *TaskScheduler* and *TaskManager* schedule and execute tasks in a multi-threading manner; they will be discussed in detail in Part B of this section. A task is defined as any type of user commands such as opening a data file, generating a design, creating a metamodel, or performing optimization. Class *TaskPanel* in Package *HiPPO.TaskPanels* is the base class from which a UI panel in each of the four sub-packages is derived. For example, Class *DOEPanel* in Package *HiPPO.TaskPanels.DOE* is derived from Class *TaskPanel* to provide the UI panel for DOE. Other classes in the four sub-packages perform the actual tasks. We summarize the functionalities of all classes in Table 1.

We use the unified modeling language (UML) to describe class relationships;[9] such descriptions are typically organized and given in a class diagram as shown in Fig. 4. Figure 4 only shows the major components in HiPPO due to limited space. Package *HiPPO.TaskPanels.WRT* is not shown in Fig. 4; however, it can be described in the same manner as that for Package *HiPPO.TaskPanels.DOE*.

In Fig. 4, we represent a class with a rectangle that is divided into three parts, with the top giving the class name, the middle the data of this class, and the bottom the operations on the data of this class. The solid line linking two classes indicates class usage, with one pointed to by an arrow using the other one. Note that an arrow is shared by different pairs of classes if arrows from different pairs point to the same class. For instance, the arrow pointing

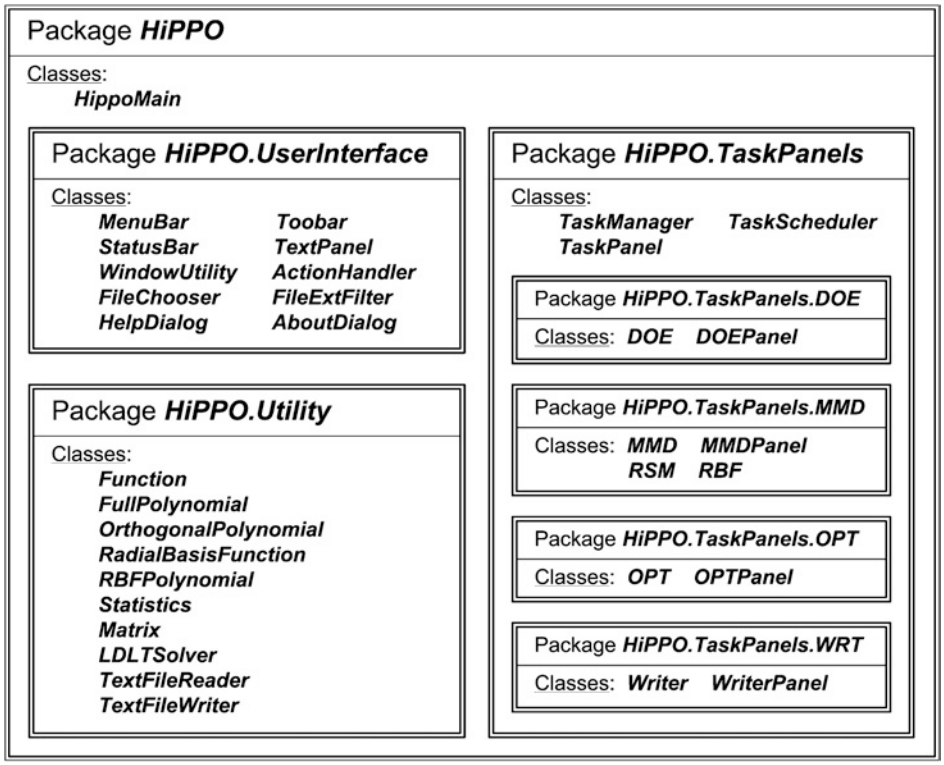

Fig. 3 Class organization in HiPPO.

to Class *TaskPanel* is shared by the three classes, *DOEPanel*, *MMDPanel*, and *OPTPanel*. The dotted line between two classes indicates class inheritance, with one pointed to by the arrow being the base class and the other the derived class. The arrow on a dotted line can also be shared by different pairs of classes if they all have the same base class.

The program starts from Class *HippoMain* in which the UI components such as the menu bar, toolbar, and main panel are created. Class *TaskManager*, which is also created in Class *HipppoMain*, creates the four UI panels (the UI panel for Package *HiPPO.TaskPanels.WRT* is not shown in Fig. 4) that are all derived from the base class *TaskPanel*. The abstract class *Function* is the base for all types of functions; both polynomials and radial basis functions are derived from it. Classes *RSM* and *RBF* implement the response surface methodology and radial basis functions, respectively; they are used in the general metamodeling class *MMD* that is used in *MMDPanel*. Classes *DOE*, *OPT*, and *Writer* are used in the same way as Class *MMD*.

Once the program starts, it waits for a user command to execute a task. All tasks are queued and executed by the task scheduler in a multi-threading fashion. As a consequence, the UI is not locked and a new task can be issued by the user before the existing tasks are finished. Details of multi-threading design are given next.

B. Multi-threading Design in HiPPO

A user program is executed in a process created by the operating system. Without multi-threading, there is only one flow of execution that uses all the CPU time allocated to the process. This is fine for most computation-intensive programs but causes problems with interactive programs such as GUI applications. In a GUI application executed in a single-thread process, when a user command is issued from the UI, the flow of execution is dedicated to fulfill the task. As a consequence, the UI stops responding until the task is finished. This interface-locking problem definitely needs to be solved in an integrated OF such as HiPPO, because performing a multi-objective optimization task may take a long time to finish.

Table 1 Functionalities of classes in HiPPO

Package	Class	Function description
HiPPO	*HippoMain*	Program entry for GUI setup
HiPPO.UserInterface	*MenuBar*	System menu component
	ToolBar	System toolbar
	StatusBar	Task execution status
	TextPanel	Display of text data
	WindowUtility	Window style selection
	ActionHandler	User action capturing and handling
	FileChooser	File open/save dialogs
	FileExtFilter	File filters using file extension names
	HelpDialog	The Help dialog
	AboutDialog	The About dialog
HiPPO.Utility	*Function*	An abstract class defining interface methods for all types of mathematical functions
	FullPolynomial	Complete polynomial functions
	OrthogonalPolynomial	Orthogonal polynomial functions
	RadialBasisFunction	Radial basis functions
	RBFPolynomial	Polynomial functions used in radial basis functions
	Statistics	Statistical analysis on regression functions
	Matrix	Matrix manipulations
	LDLTSolver	The matrix LDLT solver
	TextFileReader	A general-purpose text file reader
	TextFileWriter	A general-purpose text file writer
HiPPO.TaskPanels	*TaskManager*	Management of task threads and task submission
	TaskScheduler	A scheduler for task execution
	TaskPanel	An abstract class defining features and interface methods for all task panels (interfaces)
HiPPO.TaskPanels.DOE	*DOEPanel*	Interface panel for design of experiments (DOE)
	DOE	Class for performing DOE tasks
HiPPO.TaskPanels.MMD	*MMDPanel*	Interface panel for metamodeling (MMD)
	MMD	Class for performing MMD tasks
	RBF	Metamodeling with radial basis functions (RBF)
	RSM	Metamodeling with response surface methodology (RSM)
HiPPO.TaskPanels.OPT	*OPTPanel*	Interface panel for optimization (OPT)
	OPT	Class for performing OPT tasks
HiPPO.TaskPanels.WRT	*WriterPanel*	Interface panel for a text editor
	Writer	Class for performing text editing

In simple words, the multi-threading mechanism uses multiple independent flows of executions within a process; each flow is called a thread. A time-sharing mechanism is adopted among all threads similar to the time-sharing used by an operating system. In the multi-threading design, we execute user tasks in different threads from that of the UI. Therefore, the UI can respond and accept more user commands even when there are user tasks being executed. However, we need to set a limit to the number of threads that can run simultaneously; too many concurrent threads cause performance degradation. To be able to accept user commands from the UI even when this limit is reached, we use a queue to store user requests and use a scheduler to execute queued tasks when a thread becomes available. The design of the multi-threading task manager is shown Fig. 5.

The task manager runs in the main thread of the process; it creates the UI panels in the main thread and a task scheduler in another separate thread. The task scheduler frequently checks the task queue and executes the tasks in a first-in-first-out manner if there are available threads. The scheduler goes into sleep mode to save CPU time if there is no task in the queue or no available thread. If a UI panel receives a user command, it puts the requested task into the task queue and continues to wait for user input. The UI is never locked in this multi-threading design.

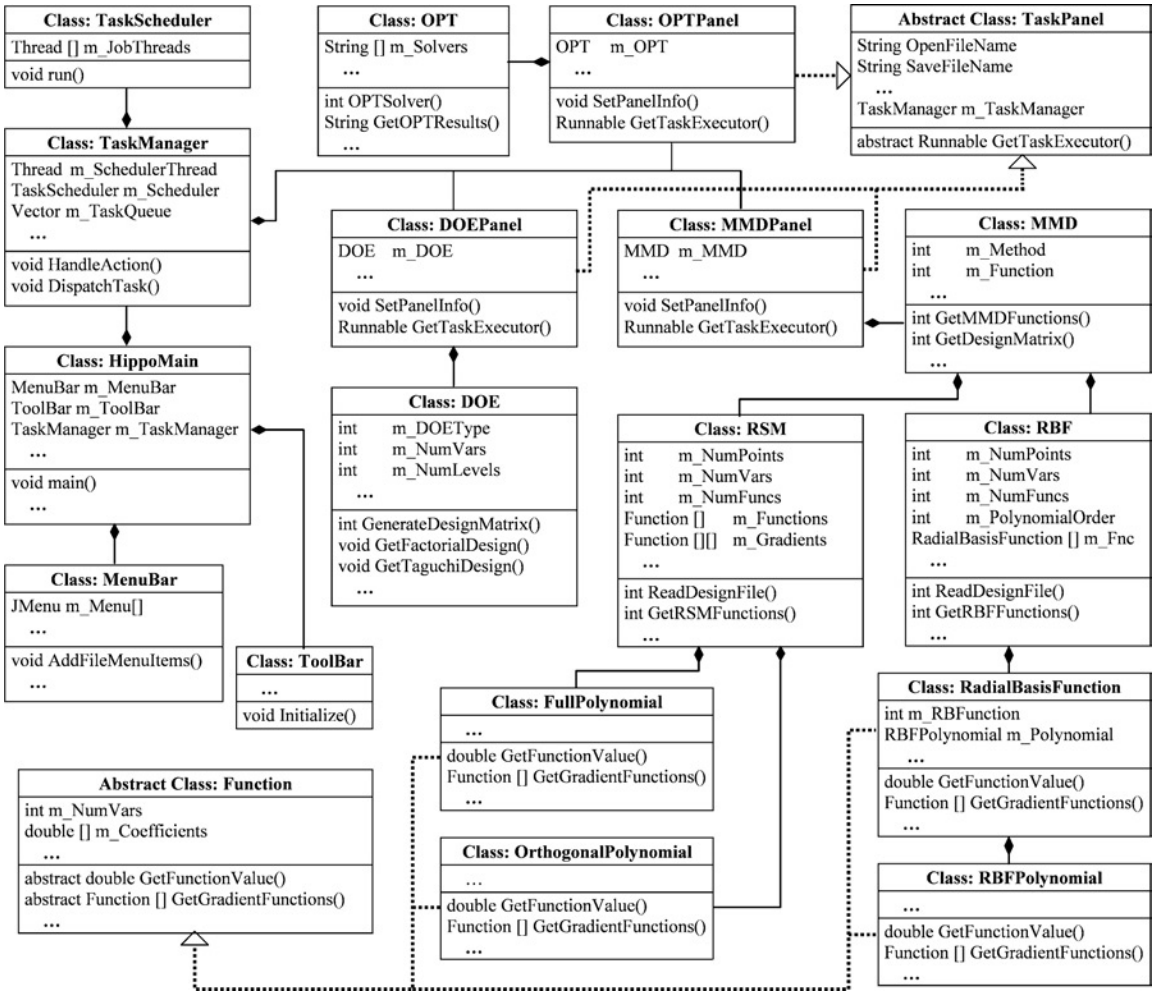

Fig. 4 The class diagram for HiPPO.

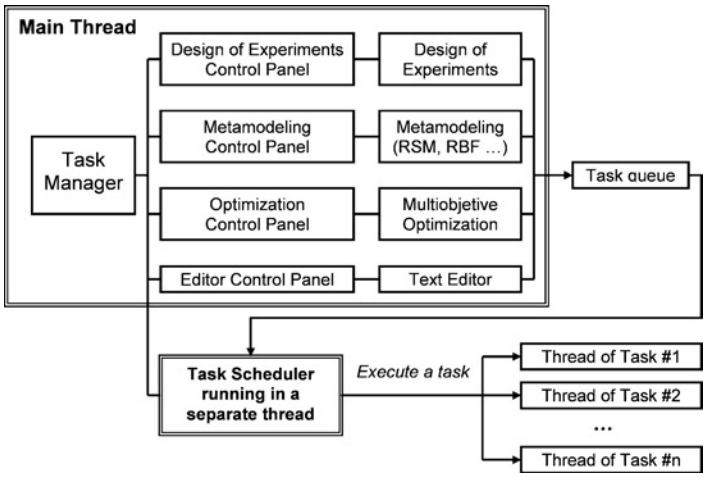

Fig. 5 The multi-threading design in HiPPO.

C. Class Organization of the GOI-based Optimizer

We give a brief overview of the GOI-based optimizer in this article; its detailed design, algorithms, and performance evaluation are discussed in a separate article.[10] The GOI-based optimizer is an object-oriented program for multi-objective optimization. This standalone system is incorporated into HiPPO and can be started from HiPPO's UI. We give the class diagram of this optimizer in Fig. 6. The rectangles with names "Program entry", "FSQP Solver", and "RFSQP Solver" are not classes, but we treat them as classes in Fig. 6 for ease of discussion.

In Fig. 6, "Program entry" is the main function of this GOI-based optimizer. "FSQP Solver" and "RFSQP Solver" are two existing optimization solvers (also called optimizer) that were developed by Laurence, et al. implementing

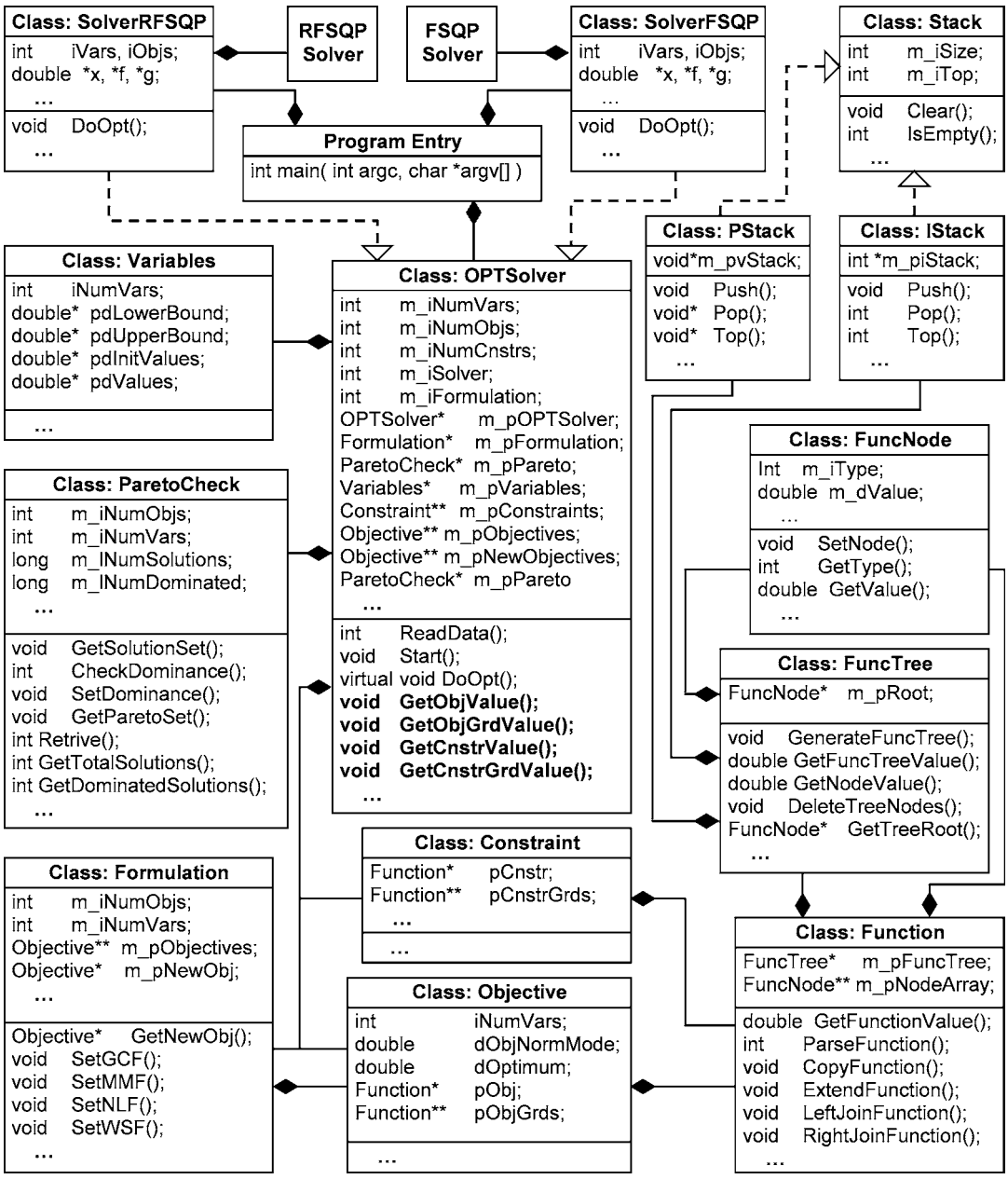

Fig. 6 The class diagram for the GOI-based multi-objective optimizer.

Table 2 Functionalities of classes and functions in the GOI-based multi-objective optimizer.

Class/Function	Functionality
Main	Program entry function
OPTSolver	The generic optimizer interface (GOI)
SoverFSQP	Specific interface for the FSQP optimizer
SoverRFSQP	Specific interface for the RFSQP optimizer
Function	Internal function representation (IFR)
Formulation	Function formulations using IFRs for multi-objective optimizations
FuncTree	Binary tree representation for IFR
FuncNode	A node used in class *FuncTree*
Objective	Objective functions and their gradient functions in IFR format
Constraint	Constraint functions and their gradient functions in IFR format
Variables	Design variables
Stack	A base class for all types of stacks
Istack	A stack for integers
PStack	A stack for pointers
ParetoCheck	Obtaining solutions on the Pareto Frontier by Pareto non-dominance check
FSQP Solver	Functions of the FSQP optimizer
RFSQP Solver	Functions of the RFSQP optimizer

the feasible sequential quadratic programming method.[11] Classes *SolverFSQP* and *SolverRFSQP* define the special interfaces to the two optimizers, respectively; both classes are derived from Class *OPTSolver* that is the major component of the GOI.

In Class *OPTSolver*, the four functions *GetObjValue*, *GetObjGrdValue*, *GetCnstrValue*, and *GetCnstrGrdValue* are used to obtain values of objectives, objective gradients, constraints, and constraint gradients, respectively. These four functions provide a generic interface to the optimization problems, because all of the functions are in the IFR format no matter which optimization solver is actually used (see Fig. 2). On the other hand, the four functions are also generic to the various optimization solvers, because all of the solvers will call these four functions to obtain function and gradient values no matter what the optimization problem is. Note that the two functions for calculating gradients are only needed for gradient-based optimizers.

All input functions are converted into IFR formats given by Class *Function*. We can form a new objective function in Class *Formulation* if a formulation (i.e., weighted sum formulation) is specified for multi-objective optimization. The input objective functions and their gradients are defined in Class *Objective*, and the input constraint functions and their gradients are defined in Class *Constraint*. Class *ParetoCheck* performs the Pareto non-dominance check for multi-objective optimization problems. A summary of class functionalities is given in Table 2.

To add a new optimizer, we can simply derive a class from class *OPTSolver* to define a new interface (see Fig. 2). All other components and input functions for existing problems remain unchanged.

IV. System Implementation

There are two commonly used OO programming languages, Java[12] and C++;[13] both languages can be used to implement HiPPO and the GOI-based optimizer. Since HiPPO is designed as a GUI application, it is advantageous to use Java for implementation. Java provides abundant standard classes to support various types of development including GUI and multi-threading applications, all in a platform independent manner. This platform-independent GUI is highly desirable, because it allows the application to run on different operating systems such as Unix, Linux, Mac, and Windows 98/2000/XP without the need of recoding and program recompilation. This feature is not currently supported in any other programming languages. Based on these reasons, we selected Java to implement HiPPO.

Figure 7 shows the optimization UI of HiPPO running on a Windows XP system. The UIs for DOE and metamodeling are similar to that of Fig. 7, except for the displayed contents. The main panel in Fig. 7 displays the output of the optimizer, the panel at the top-right corner gives the user options for optimization, and the panel at the bottom-right

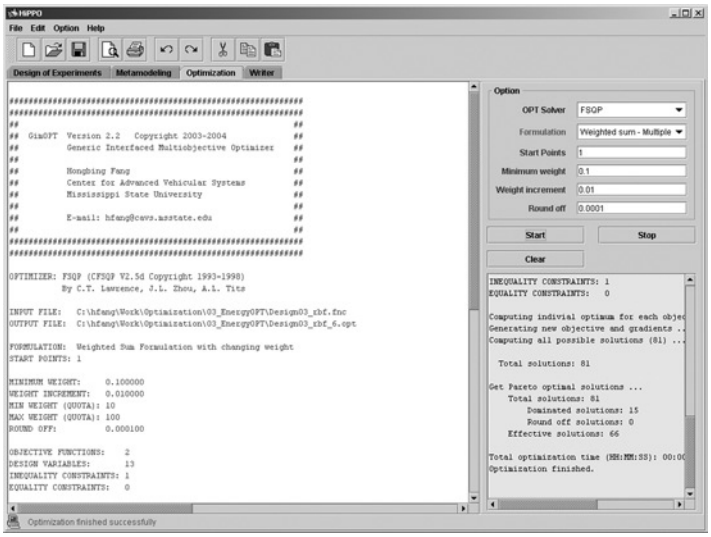

Fig. 7 HiPPO's user interface for optimization.

corner shows the statuses of all executing tasks. With the multi-threading design, these UIs are not locked at any time. For example, we can continuously submit multiple optimization tasks in the optimization UI and/or switch to the UI of DOE or metamodeling to start a new task.

We selected the C++ programming language for implementing the GOI-based optimizer, because using C++ made it easy to interface with existing optimizers that were mostly developed in Fortran, C, and C++. Although Java can also be used instead of C++, the interfacing with other languages in Java is more complicated. In addition, the GOI-based optimizer is not a GUI application and will not take advantage of Java's support for GUI development. Furthermore, Java's platform-independent feature is lost when a Java program is combined with other programs developed in Fortran, C, or C++. Therefore, C++ was a better choice for the GOI-based optimizer. Although the GOI-based optimizer is not platform independent, a simple recompilation is sufficient before running on a new system, because it does not require graphics support.

The GOI-based optimizer is a standalone system and can also be started in HiPPO. The output in Fig. 7 is actually from an optimization task performed by the GOI-based optimizer.

V. Examples of Design Optimization using HiPPO

We use two examples in this section to illustrate how HiPPO and the GOI-based optimizer can be used for various optimization problems. The first example is a multi-objective optimization problem to improve a vehicle's crashworthiness design, while the second example is a widely used benchmark problem.

A. Multi-objective Crashworthiness Optimization

In this simulation-based design optimization problem, we optimized the design of a 1996 Dodge Neon to improve its crashworthiness in an offset-frontal impact (OFI) and side impact (SI). The finite element model was originally developed for full-frontal impact simulations at the National Crash Analysis Center in the US.[14,15] We modified and updated the model so that it could be used in OFI and SI, as shown in Figs. 8a and 8b, respectively.

Based on an analysis of the energy absorption histories in both impacts, we selected twenty-one components and used their thickness as design variables. We only needed 13 design variables due to component symmetry. The three design objectives are maximizing the energy absorption in OFI, minimizing the intrusion distances in SI, and minimizing the mass of the selected components. The weighted sum formulation of this multi-objective optimization

Fig. 8 Finite element models for offset-frontal and side impacts.

problem is given as

$$\begin{aligned} &\text{Min } F(x) = -W_1 f_1(x) + W_2 f_2(x) + W_3 f_3(x) \\ &\text{s.t. } x_i^l \le x_i \le x_i^u \qquad i = 1, 13 \\ &\qquad W_j > 0, \quad \sum W_j = 1, \ \ j = 1, 3 \end{aligned} \tag{1}$$

where $f_i(x)$ $(i = 1, 2, 3)$ is the i^{th} objective function and W_i $(i = 1, 2, 3)$ represents the weight coefficient for the i^{th} objective. Each weight is greater than zero and the sum of all weights equals one. The negative form of $f_1(x)$ is used, because this function is to be maximized. Larger weights indicate a greater importance of the objective function, and the optimum design will be relatively closer to the optimum of the objective with a larger weight.

Our first task was to create metamodels for the first two objectives. Due to the extremely high cost of crash simulations, we selected the Taguchi orthogonal array L_{27} to do sampling that only requires 27 simulations for 13 design variables.[16] The normalized design matrix was generated with HiPPO and given in Appendix A-1 (lines No. 1 to 27 in the matrix). Line No. 0 in the matrix corresponds to the original simulation and is not part of the Taguchi array. We performed simulations corresponding to each design and the results are also given in Appendix A-1 under the two columns of $f_1(x)$ and $f_2(x)$.

With the design matrix and corresponding values of the responses, we created metamodels for the two objectives using the response surface methodology provided by HiPPO. The generated objective functions and their gradients were saved in an input file as shown in Appendix A-2. Function $f_3(x)$ was not generated in HiPPO, because we knew its explicit format. The input file also has gradients of all the three functions.

Finally, we solve this multi-objective optimization problem using the GOI-based optimizer with the weighted sum formulation. In HiPPO, we can simply specify a minimum value for all the weight coefficients and a weight increment; the GOI-based optimizer automatically computes optima for all combinations of weight coefficients and generates the Pareto frontier by performing a Pareto non-dominance check. In this example, we used a minimum weight of 0.01 and a weight increment of 0.002; we obtained 8,327 solutions on the Pareto frontier from a total of 118,341 solutions. An excerpt from the output file of the GOI-based optimizer is given in Appendix A-3.

A final solution is selected from the Pareto optimum set and validated with finite element simulations.[17] This solution represents no change to $f_1(x)$, 6% reduction on $f_2(x)$, and 14.5% reduction on $f_3(x)$. Errors of the metamodels for $f_1(x)$ and $f_2(x)$ are only 1.9% and 0.86%, respectively. There is no error on $f_3(x)$ because we have the exact function. Details of optimization results are not discussed here, since that is beyond the scope of this article.

B. An Optimization Benchmark

Our second example is a benchmark problem in which we need to find the minimum of a single objective function with both equality and inequality constraints. This problem is given as

$$\begin{aligned} &\text{Min } f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)\, x_2^3 \\ &\text{s.t. } g_1(x) = -x_4 + x_3 - 0.55 \le 0 \\ &\qquad g_2(x) = -x_3 + x_4 - 0.55 \le 0 \end{aligned}$$

$$h_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$
$$h_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$
$$h_5(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 - x_1 = 0$$
$$0 \le x_1 \le 1200, 0 \le x_2 \le 1200, -0.55 \le x_3 \le 0.55, -0.55 \le x_4 \le 0.55 \quad (2)$$

The best known solution to this problem is $x^* = [679.9453, 1026.067, 0.1188764, -0.3962336]$ and $f(x^*) = 5126.4981$.[18] We generated the input file for this problem as shown in Appendix A-4 and used HiPPO to solve this problem; an excerpt from the optimization output is given in Appendix A-5. The results under "OPTIMAL SOLUTION" in the output have the same values for design variables and objective as those from the literature.

VI. Conclusion

In this study, we developed a general-purpose design optimization framework using the object-oriented design concept. This framework integrates design of experiments, metamodeling, and multi-objective optimization, all under a graphical user interface and supported by the multi-threading mechanism. By designing a generic optimizer interface (GOI) in the multi-objective optimizer, various types of existing optimization routines can be easily incorporated without the need for major modification. On the other hand, any optimization routines in GOI-based optimizer can be used to solve arbitrary user-provided optimization problems without the need for user programming and program recompilation.

This optimization framework has been successfully used in various studies including metamodeling-based design optimization, accuracy evaluation of metamodeling methods, optimization of benchmark problems, and comparison of optimization methods/routines. By incorporating more optimization routines, this framework will be very useful in evaluating the efficiency and convergence of various optimization methods for single- and multi-objective optimization using benchmarks and real application problems, both new and already defined for the GOI-based optimizer.

Appendix

A-1. Design Matrix for the Vehicle Crashworthiness Problem

```
#################################################################
#    DESIGN OF EXPERIMENTS
#        HiPPO—High Performance Processing Option
#        Hongbing Fang, CAVS, Mississippi State University
#################################################################
#
# TAGUCHI DESIGN
#         Number of Data Points:         28
#         Number of Variables:           13
#         Number of Design Levels:       3
#         Number of Functions:           3
#
# NUM_POINTS NUM_VARS NUM_LEVELS NUM_FUNCS
       28       13         3         3
#
#No    X1  X2  X3  X4  X5  X6  X7  X8  X9  X10 X11 X12 X13 f1(X)    f2(X)
#-------------------------------------------------------------------
0   0   0   0   0   0   0   0   0   0   0   0   0   0      3.0794   370.0719
1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1      3.1304   385.2242
2  -1  -1  -1  -1   0   0   0   0   0   0   0   0   0      3.0488   369.5117
3  -1  -1  -1  -1   1   1   1   1   1   1   1   1   1      2.8959   355.3133
```

4	-1	0	0	0	-1	-1	-1	0	0	0	1	1	1	2.9877	382.9333
5	-1	0	0	0	0	0	0	1	1	1	-1	-1	-1	3.0183	363.6992
6	-1	0	0	0	1	1	1	-1	-1	-1	0	0	0	3.0284	377.7200
7	-1	1	1	1	-1	-1	-1	1	1	1	0	0	0	2.9673	366.2567
8	-1	1	1	1	0	0	0	-1	-1	-1	1	1	1	3.2426	384.3650
9	-1	1	1	1	1	1	1	0	0	0	-1	-1	-1	3.0183	363.8667
10	0	-1	0	1	-1	0	1	-1	0	1	-1	0	1	3.0183	375.6625
11	0	-1	0	1	0	1	-1	0	1	-1	0	1	-1	3.0488	361.6667
12	0	-1	0	1	1	-1	0	1	-1	0	1	-1	0	3.0081	337.1767
13	0	0	1	-1	-1	0	1	0	1	-1	1	-1	0	3.2324	375.2258
14	0	0	1	-1	0	1	-1	1	-1	0	-1	0	1	2.9877	360.6892
15	0	0	1	-1	1	-1	0	-1	0	1	0	1	-1	2.9673	375.8700
16	0	1	-1	0	-1	0	1	1	-1	0	0	1	-1	3.0488	365.3567
17	0	1	-1	0	0	1	-1	-1	0	1	1	-1	0	2.8755	377.9825
18	0	1	-1	0	1	-1	0	0	1	-1	-1	0	1	3.1610	357.4850
19	1	-1	1	0	-1	1	0	-1	1	0	-1	1	0	3.0081	370.7067
20	1	-1	1	0	0	-1	1	0	-1	1	0	-1	1	2.9775	354.4508
21	1	-1	1	0	1	0	-1	1	0	-1	1	0	-1	3.2222	339.8133
22	1	0	-1	1	-1	1	0	0	-1	1	1	0	-1	2.8755	366.9275
23	1	0	-1	1	0	-1	1	1	0	-1	-1	1	0	3.2018	345.2942
24	1	0	-1	1	1	0	-1	-1	1	0	0	-1	1	3.0794	365.0383
25	1	1	0	-1	-1	1	0	1	0	-1	0	-1	1	3.0488	361.0067
26	1	1	0	-1	0	-1	1	-1	1	0	1	0	-1	2.9979	373.4467
27	1	1	0	-1	1	0	-1	0	-1	1	-1	1	0	3.0183	354.9467

A-2. Optimization Input for the Vehicle Crashworthiness Problem

```
# Number of Variables, Objectives, Inequality and Equality Constraints
#-------------------------------------------------------------
Variables:         13
Objectives:        3
IneqConstraints:   0
EqConstraints:     0
# Objective Functions
#-------------------------------------------------------------
f1 = + 3.03 – 0.04*x1 + 0.14*x2 – 1.4*x3 –0.08*x4 + 0.34*x5 + 1.49*x6
   + 0.19*x7 + 0.08*x8 + 0.12*x9 – 0.69*x10 – 0.53*x11 –0.37*x12
   + 0.07*x13 + 0.04*x1^ 2 – 0.08*x2^ 2 + 1.16*x3^ 2 + 0.03*x4^ 2
   – 0.22*x5^ 2 – 0.4*x6^ 2 – 0.11*x7^ 2 – 0.04*x8^ 2– 0.04*x9^ 2
   + 0.13*x10^ 2 + 0.16*x11^ 2 + 0.27*x12^ 2 – 0.01*x13^ 2;
f2 = + 374.23 – 38.44*x1 + 176.51*x2 + 0.51*x3 – 0.18*x4 –17.43*x5
   + 16.45*x6 – 14.53*x7 –59.75*x8 + 12.94*x9 + 1.51*x10 + 32.24*x11
   – 2.11*x12 – 27.55*x13 + 4.51*x1^ 2 –97.75*x2^ 2 + 0.43*x3^ 2
   – 1.87*x4^ 2 – 14.87*x5^ 2 –3.69*x6^ 2 + 8.47*x7^ 2 + 5.09*x8^ 2
   – 4.15*x9^ 2 – 0.3*x10^  2 –9.84*x11^ 2 + 4.31*x12^ 2 + 5.3*x13^ 2;
f3 = 2*0.004625*x1/0.846 + 2*0.003965*x2/0.827 + 0.004562*x3/0.647
   + 2*0.003466*x4/1.611 + 2*0.001374*x5/0.706 + 0.006665*x6/1.956
   + 0.013140*x7/0.705 + 2*0.011612*x8/0.829 + 2*0.002328*x9/1.524
   + 2*0.003814*x10/1.895  +  2*0.001972*x11/1.522 + 0.00173*x12/0.71
   + 0.004367*x13/2.606;
# Gradient Functions of the objectives
#-------------------------------------------------------------
```

```
gf1 – x1 = – 0.04 + 0.07*x1;
   …
gf1 – x13 = + 0.07 – 0.025*x13;
gf2 – x1 = – 38.44 + 9.03*x1;
   …
gf2 – x13 = – 27.55 + 10.61*x13;
gf3 – x1 = 2*0.004625/0.846;
   …
gf3 – x13 = 0.004367/2.606;
# Lower bound, upper bound, and initial values of variables
#-------------------------------------------------------------
0.6345      1.0575      0.7191
0.62025     1.03375     0.70295
0.48525     0.80875     0.54995
1.20825     2.01375     1.36935
0.5295      0.8825      0.6001
1.467       2.445       1.6626
0.52875     0.88125     0.59925
0.62175     1.03625     0.70465
1.143       1.905       1.2954
1.42125     2.36875     1.61075
1.1415      1.9025      1.2937
0.5325      0.8875      0.6035
1.9545      3.2575      2.2151
```

A-3. Optimization Results for the Vehicle Crashworthiness Problem

```
############################################################
## GimOPT Version 2.2 Copyright 2003-2004                 ##
##         Generic Interfaced Multiobjective Optimizer    ##
############################################################
OPTIMIZER: FSQP (CFSQP V2.5d Copyright 1993-1998)
           By C.T. Lawrence, J.L. Zhou, A.L. Tits

INPUT FILE:      SIintOFIengMass_qp.fnc
OUTPUT FILE:     SIintOFIengMass_qp_6.opt

FORMULATION:     Weighted Sum Formulation with changing weight
START POINTS:    100

MINIMUM WEIGHT:          0.010000
WEIGHT INCREMENT:        0.002000
MIN WEIGHT (QUOTA):      5
MAX WEIGHT (QUOTA):      500
ROUND OFF:               0.000000
OBJECTIVE FUNCTIONS:     3
DESIGN VARIABLES:        13
INEQUALITY CONSTRAINTS:  0
EQUALITY CONSTRAINTS:    0
```

```
INITIAL VALUES
-------------------------------------------------------------
F1             F2             F3             X1             X2             …
-------------------------------------------------------------
3.101339e+00   3.794394e+02   8.226300e-02   7.191000e-01   7.029500e-01   …

INDIVIDUAL OPTIMUM IN SINGLE-OBJECTIVE OPTIMIZATION
-------------------------------------------------------------
F1             F2             F3             X1             X2             …
-------------------------------------------------------------
3.301586e+00 N/A             N/A           1.057500e+00     9.602824e-01  …
N/A          3.288625e+02    N/A           1.057500e+00     6.202500e-01  …
N/A          N/A             7.937385e-02  6.345000e-01     6.202500e-01  …
TOTAL SOLUTIONS:     118341
DOMINATED SOLUTIONS: 110014
ROUND OFF SOLUTIONS: 0
EFFECTIVE SOLUTIONS: 8327
OPTIMAL SOLUTION SET
-------------------------------------------------------------
F1             F2             F3             X1             X2             …
-------------------------------------------------------------
3.136557e+00   3.704600e+02   7.937456e-02   6.345000e-01   6.202500e-01 …
3.136692e+00   3.704481e+02   7.937464e-02   6.345000e-01   6.202500e-01 …
     …
3.225304e+00   3.348122e+02   9.319364e-02   1.057500e+00   6.202500e-01 …
3.225742e+00   3.347218e+02   9.326681e-02   1.057500e+00   6.202500e-01 …
END_OF_OPTIMIZATION
```

A-4. Optimization Input for the Benchmark Problem

```
# Number of Variables, Objectives, Inequality and Equality Constraints
Variables:         4
Objectives:        1
IneqConstraints:   2
EqConstraints:     3

# Objective functions
f1 = 3*x1 + 0.000001*x1^ 3 + 2*x2 + (0.000002 / 3)*x2^ 3;

# Constraint functions
c1 = x3 - x4 - 0.55;
c2 = -x3+x4 - 0.55;
c3 = 1000*sin(-x3 - 0.25) + 1000*sin(-x4 - 0.25) - x1 + 894.8;
c4 = 1000*sin(x3 - 0.25) + 1000*sin(x3 - x4 - 0.25) - x2 + 894.8;
c5 = 1000*sin(x4 - 0.25) + 1000*sin(x4 - x3 - 0.25) + 1294.8;

# Gradient functions of the objectives
gf1-x1 = 3 + 0.000003*x1^ 2;
gf1-x2 = 2 + 0.000002*x2^ 2;
gf1-x3 = 0;
gf1-x4 = 0;
```

```
# Gradient functions of the constraints
gc1-x1 = 0;
gc1-x2 = 0;
gc1-x3 = 1;
gc1-x4 = -1;
gc2-x1 = 0;
gc2-x2 = 0;
gc2-x3 = -1;
gc2-x4 = 1;
gc3-x1 = -1;
gc3-x2 = 0;
gc3-x3 = -1000*cos(-x3 - 0.25);
gc3-x4 = -1000*cos(-x4 - 0.25);
gc4-x1 = 0;
gc4-x2 = -1;
gc4-x3 = 1000*cos(x3 - 0.25) + 1000*cos(x3 - x4 - 0.25);
gc4-x4 = -1000*cos(x3 - x4 - 0.25);
gc5-x1 = 0;
gc5-x2 = 0;
gc5-x3 = -1000*cos(x4 - x3 - 0.25);
gc5-x4 = 1000*cos(x4 - 0.25) + 1000*cos(x4 - x3 - 0.25);

# Lower bound, upper bound, and initial values of variables
0   1200    73
0   1200    117
-0.55   0.55    -0.2
-0.55   0.55    0.3
# End of Function File
```

A-5. Optimization Results for the Benchmark Problem

```
############################################################
## GimOPT Version 2.2 Copyright 2003-2004                 ##
##         Generic Interfaced Multiobjective Optimizer    ##
############################################################
OPTIMIZER: FSQP (CFSQP V2.5d Copyright 1993-1998)
            By C.T. Lawrence, J.L. Zhou, A.L. Tits

INPUT FILE:    C:\G-Suite_Test_Functions\GSuiteTestFunc05.fnc
OUTPUT FILE:   C:\G-Suite_Test_Functions\GSuiteTestFunc05_1.opt

FORMULATION:   No formulation
START POINTS:  1

MINIMUM WEIGHT:     N/A
WEIGHT INCREMENT:   N/A
ROUND OFF:          N/A

OBJECTIVE FUNCTIONS:     1
DESIGN VARIABLES:        4
INEQUALITY CONSTRAINTS:  2
EQUALITY CONSTRAINTS:    3
```

```
INITIAL VALUES
-------------------------------------------------------------
F1             X1             X2             X3             X4
-------------------------------------------------------------
4.544568e+002  7.300000e+001  1.170000e+002  -2.000000e-001  3.000000e-001

INDIVIDUAL OPTIMUM IN SINGLE-OBJECTIVE OPTIMIZATION
-------------------------------------------------------------
F1             X1             X2             X3             X4
-------------------------------------------------------------
5.126498e+003  6.799453e+002  1.026067e+003  1.188764e-001  -3.962336e-001

OPTIMAL SOLUTION
-------------------------------------------------------------
F1             X1             X2             X3             X4
-------------------------------------------------------------
5.126498e+003  6.799453e+002  1.026067e+003   1.188764e-001 -3.962336e-001

END_OF_OPTIMIZATION
```

References

[1]Montgomery, D. C., *Design and Analysis of Experiments*, John Wiley & Sons, Inc., New York, 2001, Chaps. 10, 11.

[2]Hardy, R. L., "Multiquadratic Equations of Topography and Other Irregular Surfaces," *Journal of Geophysics*, Vol. 76, 1971, pp. 1905–1915.

[3]Balabanov, V. O., Charpentier, C., Ghosh, D., Quinn, G., Vanderplaats, G. N., and Venter, G. "VisualDOC: A Software System for General-purpose Integration and Design Optimization," *The 9th AIAA/ISSMO on Multidisciplinary Analysis and Optimization Conference*, Atlanta, GA, 2002.

[4]ISIGHT, Engineering Design Improvement Software, Engineous Software Inc., Morrisville, NC, 2002.

[5]OptiStruct, Finite Element-based Optimization Tool, Altair Engineering, Troy, MI, 2003.

[6]Stander, N., Eggleston, T., Craig, K., and Roux, W., "Design Optimization Software for the Engineering Analyst, LS-OPT User's Manual," Livermore Software Technology Corporation, CA, 2003.

[7]Haug, E., "PAM-OPT Solver-Reference manual," Pam-System International, MI, 2000.

[8]Wolpert, D. H. and Macready, W. G., "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 1, 1977, pp. 67–82.

[9]Rumbaugh, J., Jacobson, I., and Booch, G., "*The Unified Modeling Language Reference Manual*," 2nd edition, Addison-Wesley Professional, Boston, MA, 2004.

[10]Fang, H. and Horstemeyer, M. F., "A Generic Optimizer Interface for Programming-free Optimization Systems," *Advances in Engineering Software*, (to be published).

[11]Lawrence, C. T., Zhou, J. L., and Tits, A. L., "User's Guide for CFSQP Version 2.5," Electrical Engineering Department and Institute for Systems Research, University of Maryland, College Park, MD, 1997.

[12]Schildt, H., "*Java 2: The Complete Reference*," McGraw-Hill/Osborne Media, Emeryville, CA, 2002.

[13]Lafore, R., "*Object-oriented Programming in C++*," Sams Publishing, Indianapolis, IN, 1999.

[14]Zaouk, A. K., Marzougui, D., and Bedewi, N. E., "Development of A Detailed Vehicle Finite Element Model, Part I: Methodology," *International Journal of Crashworthiness*, Vol. 5, No. 1, 2000, pp. 25–35.

[15]Zaouk, A. K., Marzougui, D., and Kan, C. D., "Development of A Detailed Vehicle Finite Element Model, Part II: Material Characterization and Component Testing," *International Journal of Crashworthiness*, Vol. 5, No. 1, 2000, pp. 37–50.

[16]Taguchi, G., "*Taguchi Method—Design of Experiments, Quality Engineering Series Vol. 4*," Japanese Standards Association, ASI Press, Tokyo, Japan, 1993.

[17]Fang, H., Solanki, K., and Horstemeyer, M. F., "Numerical Simulations of Multiple Vehicle Crashes and Multidisciplinary Crashworthiness Optimization," *International Journal of Crashworthiness*, Vol. 10, No. 2, 2005, pp. 161–171.

[18]Koziel, S. and Michalewicz, Z., "Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization," *Evolutionary Computation*, Vol. 7, No. 1, 1999, pp. 19–44.