Simulation-based development of heuristic strategies for complex system convergence

C McCulley, K Hulme, and C L Bloebaum

Multidisciplinary Optimization and Design Engineering Laboratory (MODEL), Department of Mechanical and Aerospace Engineering, University at Buffalo, Buffalo NY

The feasibility of system-level optimization in complex engineering design rests on the capability to iteratively converge a system of coupled subprocesses in a reasonable amount of time and at a reasonable cost. Each of these subprocesses, or modules, has an individual time and cost to execute. Convergence strategies are systems of rules for executing through the set of modules. Strategies can be developed which draw information from system characteristics to reduce the overall time and cost of converging the system.

A simulator has been created which combines automatic model building and sequencing capability with an engine capable of running either sequential or parallel execution strategies. Several first generation strategies, both sequential and parallel, are explored using this technique.

INTRODUCTION

Complex system engineering, such as that undertaken for the design and production of aircraft, is a highly multidisciplinary process which is carried out as a collection of smaller independent efforts. The effect of the interactions between disciplinary or subsystem components of a design process upon the system-level characteristics of the final system is often poorly understood. In addition, the large nature of these systems have made coordinated system-wide optimization infeasible. Traditional reliance has thus been placed on trade studies of a small number of candidate designs. The improvement of this situation is the purview of the field of Multidisciplinary Design Optimization (MDO).

An MDO process, as proposed by Sobieski[12] and others, achieves system-level optimization, but at large cost caused by the required iteration. A radical increase in the efficiency of converging an iterative design or analysis process is a fundamental step required to make system-level optimization of complex systems, a highly iterative endeavor, a more attractive option. Toward this end, it is useful to gather heuristic information about the behavior of complex iterative systems within an MDO framework. Simulation is an important tool in this endeavor, allowing convergence data generation from many systems of varying characteristics, illuminating concepts that will form the basis of improvements in the convergence processes of real systems.

The optimization process (see Figure 1), begins with the design variables being set to initial values. The coupled analysis of the system defined by these variables is solved, which sets the values of *behavior variables* which are the analysis outputs. This is an iterative process internal to the optimization. Using both the design and behavior variables, the total system sensitivity is calculated from local subproblem sensitivities using the Global Sensitivity Equations (GSE) [13]. This derivative information is used by the optimizer to choose the next design point, and the process repeats if it has not converged to an optimum.

The coupled analysis is represented by a Design Structure Matrix (DSM), shown in Figure 2. The DSM was introduced by Steward[15] and adopted and extended by Rogers for DeMAID (Design Managers Aide for Intelligent Decomposition) [9]. In a DSM, boxes along the diagonal represent modules, which represent the individual subproblems into which the analysis is decomposed. The key feature of a module is the fact that it takes data as input and produces other data as output. Thus, each module can be considered, from the structural standpoint, to be a black box. Each of the modules is coupled through its input-output data to other mod-

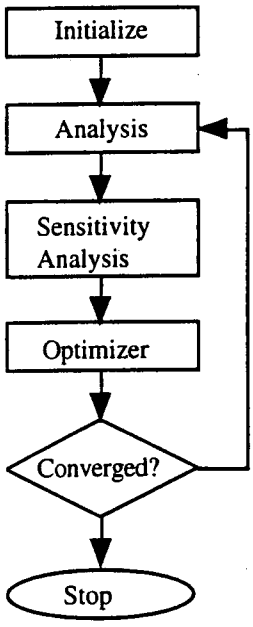

Figure 1. Optimization Procedure.

ules that either provide or require this data, respectively.

The modules along the diagonal are in a particular sequence. If one module passes information to another later in the sequence, the coupling is termed a feedforward. Feedforwards appear above the diagonal. If a module requires information from one later in the sequence, the coupling is a feedback. Feedbacks appear below the diagonal. Couplings are represented by a dot or "bullet" in the appropriate location connected to the respective modules by straight lines. A coupling exists if and only if a bullet is present, so two lines merely crossing do not have an information flow between them.

Each module can be executed with the most recent data it has received to produce updated output which may then be used by other modules. An iterative process is used to converge the modules to a solution in nonhierarchic systems where lateral couplings exist.

The analysis must be repeatedly converged at different design points to perform the optimization. For complex systems, analysis is itself large and expensive, both in terms of time and cost. The efficiency of carrying out the analysis procedure depends not only on the efficiency of the various tasks being performed in the modules, but also to a great extent on how the process is iterated. The question that arises then is how best to execute the modules to reduce the time and cost to get a converged solution to the behavior variables. Within that question arises others, such as how best to take advantage of parallelism. In real terms, the answers to such questions can help lead to better management of time and resources in complex product development processes which incorporate system-level optimization.

Simulation is an instrumental tool to help find these answers. A simulator has been created for this purpose the function of which is not unlike discrete event simulation, a technique which has been used very successfully to model and simulate industry production systems [11]. It is also finding application in many other fields. Discrete event simulation typically uses a time-slicing technique to model the advancement of processes in time. As time moves forward by discrete steps, processes begin or end, material moves between processes, control decisions are made, etc. This is very similar to the convergence process in decomposed nonhierarchic design processes. Each subproblem can begin or end, and requires a certain finite time to run. As they complete, data moves between them. There are also some key differences. Generally, a discrete event simulation of a factory system is open-ended. The factory simulation can run indefinitely in time while data is collected from the model. The analysis process being modeled here completes when the values of its output converges. Also, components of factory systems generally remain static in their input and output. A given machine always receives one particular kind of part and produces another particular kind of part from it. The subproblems that make up the complex system analysis, on the other hand, form a relationship between a changing input and output.

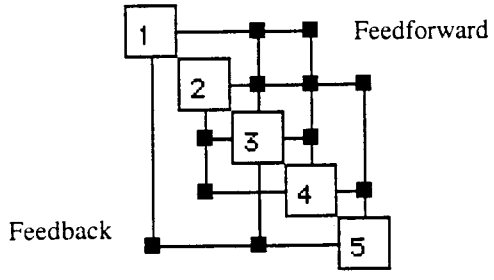

Figure 2. A design structure matrix.

Building on previous work [5], the simulator created for this study includes both model generation and the capability to sequence the DSM for selected characteristics. The result is the ability to rapidly test the effects of structure and semantics on the convergence process of a coupled system.

CONVERGENCE STRATEGIES

With large highly coupled systems, it is difficult to determine the way a system should be iterated simply by looking at the data flows. A basic convergence strategy, made up of rules which determine, based on the current state of the system, what module or modules should run, simplifies this problem and renders it capable of being automated. Numerous strategies are possible and intuitively acceptable with only two constraints. The first is that a strategy must not let the system become idle before it has converged. The second is that every module must execute. Following are descriptions of the strategies considered in this study.

Sequential Strategies

Sequential strategies require that only a single module can be in a running state at any one time.

Random. Modules are executed in a random sequence, with uniform probablility of any module being chosen next. Random execution provides a worst-case baseline. It is reasonable to expect that any useful sequential strategy should produce convergence with less time and cost than can be achieved randomly.

Straight-Through Execution (STE). The modules are executed directly in sequence starting from the top of the DSM and proceeding to the bottom. This is repeated until convergence has been achieved.

Branch at Feedbacks (BAF). In this strategy, at each feedback that is encountered, execution branches back from the source of the feedback to the module requiring the input. Innermost feedbacks are branched first. Several variations on this theme are possible. Feedbacks in between the two endpoint modules may either be ignored or internally branched. The intervening modules may be executed once, some prescribed number of times, or repeatedly until they come to an interim convergence. The branching may be ap-

plied to all feedbacks or strong feedbacks only, where strength of a feedback (or any coupling) is determined from a sensitivity analysis of the converged system. For this study, modules inside the feedback loop are executed only once.

Queue when Updated (QWU). A first-in first-out queue is maintained, and each module that receives updated information after another module has executed is queued. To avoid rampant expansion of the queue that could be caused by inputs from multiple modules, only one instance of any module may exist on the queue at any one time. A variation of this strategy not used here involves queuing a module that has received a *complete* set of updated information (i.e. all of the modules that provide input to that module have executed). This involves having an additional rule to deal with the case where no module has a complete set of information.

Parallel Strategies

Two simple parallel strategies have been implemented only for the purposes of demonstrating the concept. To take advantage of the possibilities of parallelism, far more intelligent strategies will be required.

One concern arises in parallel strategies which is not a factor in sequential execution. This is the question of when cost is incurred. For this study, cost is considered to be incurred when a module starts execution, and is incurred even if that module's completion is never achieved. This functions as a worst case scenario approximation.

Parallel Blitz (PB). Every module is started simultaneously. As each module completes, it is immediately restarted with the latest available information.

Start When Updated (SWU). The parallel cousin to QWU, this strategy runs immediately any idle module which has received updated information. If a module is already running when updated information is received, the command to restart it is held in queue until the module completes.

FACTORS AFFECTING CONVERGENCE

Numerous factors which affect the convergence of complex systems can be explored using simulation. This study will address three.

Sequencing. Sequencing has been previously explored for the reduction of convergence time and cost [5,6,8]. In certain convergence strategies, the particular sequence of modules along the diagonal of the DSM may be used to inform the procedure, directly affecting the execution order. Of the strategies above, this is the case for STE and BAF. Sequence determines the layout of the couplings over which these strategies must operate. Although STE does not directly interact with the couplings the way BAF does, the layout affects the way data is transmitted through the system during the procedure. By contrast, QWU does not depend on sequence, because the execution order is determined by the module couplings, which sequence does not affect. Neither are the two above-mentioned parallel strategies affected by sequence.

Sequencing is performed to achieve one of several objectives. Systems are sequenced for minimum feedbacks, minimum time estimate, minimum cost estimate, or some combination. Sequencing to reduce feedbacks reduces the amount of information which must be guessed during the execution process, thereby speeding convergence by providing downstream modules with more accurate inputs. It is also possible to sequence directly based on cost or time estimates for the system. This estimate involves summing for each feedback the total cost and time of all of the modules in between, and adding this result to the overall cost and time sum [9]. Direct factors such as time and cost to run individual modules of the analysis are important to the sequencing procedure because iterations may be nested in a particular convergence strategy, requiring inner loops to be executed repeatedly. The design manager may desire to find some way to move particularly expensive modules out of such loops.

Effect of Structure. There are two components to a coupled system, its structure and its semantics. Structure refers to the externally observable characteristics of the system, such as the number of modules and the way in which they are coupled together. Semantics refers to the internal content of the modules. Two systems may possess the same structure and yet have completely different semantics. Systems being many and varied, the effect of semantics on system convergence is difficult to generalize. The structure of a system is far more accessible. How much of a system's convergence behavior can be predicted based on structural considerations alone? Because the simulator can generate models to fit a given structure, structure-related convergence characteristics can be revealed by repeatedly reinstantiating the same structure. This kind of test can be run on different systems of varying structural characteristics, such as number of modules, coupling density, and sequence. This study is more limited in scope, presenting the variance caused by semantic differences in relatively few system structures.

Parallelism. The simulator makes it possible to simulate strategies which involve parallel execution of modules. Because modules have different processing times, two modules started simultaneously may not finish together, and modules may be started at any time during the convergence process. The simulator is capable of tracking this complexity and keeping the database correctly updated. This capability makes it possible to test parallel convergence strategies along with sequential ones.

SIMULATOR COMPONENTS

The simulator is made up of three integrated components: the simulator engine, CASCADE (Complex Application

Simulator for the Creation of Analytical Design Equations) [4] and Gendes [7]. The simulator engine creates the structure of the system and sets the details of the particular variable dependencies that will form a given semantic instantiation of that structure. CASCADE methodology is then applied to create a viable set of equations to form the actual semantic instantiation. Once this is successful, the simulator engine can run the system from some unconverged point to convergence using one of the strategies described above. The Gendes component can be used at any time to sequence the DSM. How or whether sequences are used depends on the convergence strategy.

The simulation process can be repeated in an automatic fashion for different strategies, different instantiations and different sequences while statistics on time and cost are collected.

Data Structure and Object Function

The simulator is object oriented and is coded in C++. Figure 3 shows the basic building blocks of its data structure. Gray boxes in the diagram are objects, referred to below in boldface. Other items are constituents of objects.

The **processor** object maintains the simulation clock and the running tally of the cost. It acts as the executive of the simulation, and is thus the storehouse of all the strategy code. Additionally, it is where the sequence information is kept.

A processor creates and acts upon a **system** object, which, as its name implies, stores all of the system-related information. This includes a **database**, a module list, and a DSM. The database object primarily keeps a list of **data** objects, each of which has a current value, an initial value, and a converged value, and which is referred to by a unique four-byte signature. The data objects also reference the module objects which create them. The system keeps a Boolean matrix which represents its DSM. The internal generation routines make certain that the semantics of the generated system match its structure correctly.

The system keeps a list of **module** objects. Each module requires inputs and produces outputs. Modules have two states, idle and running. They can be commanded by the processor to begin running, and queried for their status. Once commanded to run, a module requires a certain execution time, and incurs a certain execution cost. The module's inputs are kept as a linked list of **input tag** objects, each of which points to exactly one data object which it represents. Each module produces from one to three behavior variable outputs. This differs from the previous study [5], in which only one equation per module was allowed. Assigned outputs are kept as a linked list of **output tag** objects. Each output-tag object points to exactly one data object which it represents, and one **equation** object that produces that data. Each also contains a linked list of input tag objects, which is a subset of the module's inputs, but are separate objects also pointing to that data.

Each equation is made up of a set of **terms**, generated and adjusted with the CASCADE methodology. Each term contains a coefficient, a power, a sign, and a pointer to the data object which is the input to that term.

CASCADE Methodology

CASCADE is a standalone code programmed in FORTRAN whose methodology has been directly incorporated into the simulator objects. CASCADE is used to generate a representation of a complex system of user-specified size. It was created to provide an efficient means of determining the feasibility and robustness of MDO methodologies.

The representation that CASCADE generates is a coupled set of nonlinear analytical equations, whose output is a set of behavior variables. These equations take the form:

$$f(\mathbf{x},\mathbf{y}) = \sum_i c_i x_i^{p_i} + \sum_j d_j y_j^{q_j}$$

Where c's and d's are coefficients, p's and q's are powers, x's are the design variables which do not change over the course of the analysis, and y's are the behavior variables, which do. The CASCADE methodology, slightly modified, has been split up and made a part of the simulator's data classes so that it acts within them. The basic procedures, however, remain largely unchanged.

A set of CASCADE equations is constructed through a series of steps. First, for each term in the system, the coefficient, sign, exponent, and coupling nature (coupling to either a design variable or to a behavior variable) is determined. The initial coefficient ranges between zero and one, but is greater than zero, and the exponent is chosen to be one of 1.0, 2.0, -1.0, 0.5, 0.25, or 0.3333. Next, the magnitude of each term is determined. Design variable magnitudes are known. Because CASCADE initializes a value for each design variable (between 0.0 and 10.0), and holds that value constant throughout the convergence process, the magnitudes of design variable terms can be exactly computed on the first iteration. The magnitudes of the behavior variables are not initially known, however, and the set of equations must be

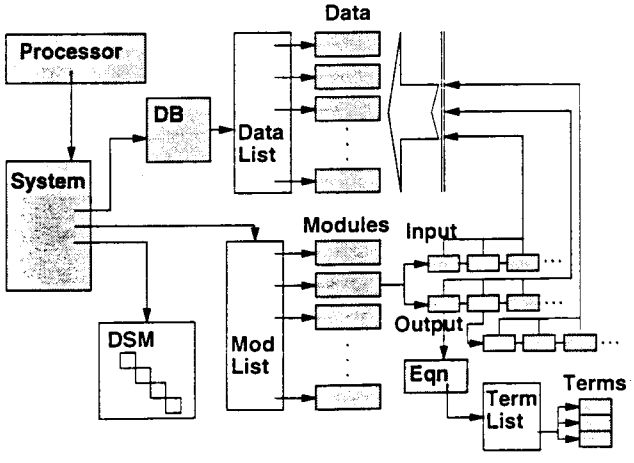

Figure 3. Simulator Data Structure.

iterated to convergence to find them. The initial guess for the magnitude of all behavior variables, on the first iteration, is chosen to be 1.0.

During this iteration process, the coefficients of the terms of the equations are altered dynamically so that a viable set of equations will result. The magnitude of every term in the system is limited to between 1.0 and 500.0 for positive terms, and between -0.25 and -100.0 for negative terms. Equation (and thus behavior variable) magnitudes are indirectly limited by this term bounding process, and as a check, are directly limited to an upper magnitude bound of 9999.0 to prevent divergence. A lower magnitude bound of 0.0 is set for every behavior variable to prevent undefined fractional exponentiations. If these bounds are violated in any iteration, corrective actions are applied. All aforementioned settings were determined heuristically based on stochastic runs of various systems.

Once the equation magnitudes have been computed and found to be within an acceptable range, a convergence check takes place. Because of the dynamic nature of the CASCADE adjustments, it is possible that the resulting values are some distance away from true convergence. Since the convergence strategy simulation relies on *a priori* knowledge of the converged value in its own convergence check, a post-CASCADE convergence step is used. This step is also used to test the viability of the system for simulation by restarting the convergence process from initial guesses some distance away (typically 30%) from the converged values. Should any of the behavior variables become negative or result in other computational problems, the instantiation is rejected. The post convergence step runs the system for a long enough time that the system will be converged well within any required precision.

System Creation and Instantiation

Each of the parts of a system are individually created. First the DSM is created randomly to a chosen system size and coupling density, with the one caveat that it must be fully coupled. That is, every module will produce output required by at least one other module, and will require input from at least one other module. Next the system is built upon that DSM. This means that the module objects are created and assigned to the system's module list. Each module is randomly given an execution time and cost which is an integer between 10 and 500. Finally the system is instantiated, which means that a set of data (design and behavior variables) and equations built upon that data is created to the specification of the existing structure. This set of data and equations forms the system's semantic component. An established structure can be reinstantiated with different sets of matching semantics.

System instantiation is a multi-step process that starts with creation of the data. First design variables are created and set with values to CASCADE bounds (0.0 to 10.0). Next, the number of equations (behavior variable outputs) for each module is chosen (between 1 and 3). For each output a data object is created and attached to the database. The initial values are set to 1.0 as per CASCADE. The output tags associated with these outputs are attached to the module. Next, inputs are assigned randomly to each module based on the DSM. For each connection, the receiving module must have at least one of the producing module's outputs as an input. Once the inputs have been assigned, each module is commanded by the system object to assign its inputs to its outputs, such that each input will be used in at least one of the module's equations. The module is then commanded to make the equations.

The creation of the equations is accomplished with the CASCADE methodology. For a given output of a given module, a term is created as previously described for each of the inputs that were assigned to the output tag. These terms are added to that output tag's equation object. When these are completely created for all modules, the system is simultaneously converged and adjusted according to the CASCADE method, and then post converged as described above. If this procedure is successful, the system, with its set of data and equations, is ready to be used in a simulation.

Gendes (Sequencing)

Gendes (from Genetic Design Sequencer) [6] operates through a genetic algorithm (GA) to determine a sequence of modules based on the characteristics of the DSM. General information on GA's can be found from Goldberg [2]. It will sequence for minimum feedbacks, minimum cost estimate, minimum time estimate, or some combination of these. Gendes was created with integration in mind, so its code remains essentially unchanged as part of the simulator.

Gendes works by coding a sequence directly into a vector by numbering the modules. For example, one such vector is [3 1 4 2 5]. The coupling information is available to the GA through the DSM for the system. A population of such sequences is processed using three operators: selection, crossover, and mutation. The selection step retains with higher probability those sequences with higher objective (fitness) values. The crossover step exchanges information between sequences to create new sequences. Gendes uses a position-based crossover method whereby randomly selected positions from the first parent sequence are selected, and any missing numbers are then filled in from the second parent in the order in which they appear in that parent. The mutation operator passes through the sequences with a small probability of swapping two positions in a sequence.

Running the Simulation

The code for any given strategy, resident in the processor object, consists of basically the same procedure. First the processor and system are reset. This means that the system clock and cost accumulator are set to zero, and each data object is reset to its initial value (typically 30% up or down from the converged value). The processor then commands one or more modules to start, and steps through time,

incrementing the system clock. At each time step each module is queried. This query offers the opportunity for a running module to determine, based on the current system time, if it should stop and update the data objects for the outputs it produces. Next the processor decides, based on the resulting system state, the DSM, the sequence or any other pertinent factors relating to the strategy being used, which module or modules (if any) should be commanded to begin running. When a module is commanded to run, it executes its equations *immediately* and stores the result in a buffer to be returned after the simulator has run through the right amount of time. This is very important, because it allows simulation of a parallel strategy to be possible. It would not be correct for a module to use results generated after it had already started running, which is possible if a shorter-running contributing module is executed or any contributing module completes execution during its run time. Thus a module must be made to use the data in the state it existed when it started.

Of course, this issue is clearer in a real system, where it actually does take a finite amount of time to produce the output data. Were new data to become available in a real system, two choices would exist: either complete the module with the data with which it was started, or start over with the new data. Exploring the results of these choices would be an interesting problem, and is certainly capable of being simulated, but for this study, the former choice is always exercised.

RESULTS

The factors of structure and sequencing are considered together for the purposes of this study. Three system structures were chosen with 10, 20 and 30 modules respectively. Each structure was instantiated five times and run with each of the sequential strategies. For STE and BAF, the strategies were run for each of the aforementioned sequences, as described in the following key used in the figures:

STE-N, BAF-N	Nominal (unsequenced) order.
STE-FB, BAF-FB	Minimum feedbacks.
STE-T, BAF-T	Minimum time estimate.
STE-C, BAF-C	Minimum cost estimate.
STE-T/C, BAF-T/C	Equal weighting on time and cost.

The modules retained their original execution time and cost values through each instantiation. The convergence time and cost results were collected for each of these runs.

Figures 4 and 5 show the time and cost results, respectively, for the ten module system. To more easily read these diagrams, keep in mind that for each instantiation the bars proceed in the same order in which the strategies appear in the legend with the random strategy to the left and QWU to the right of each subgrouping.

Despite the unrelatedness of the values of cost and time chosen for the modules, with potential differences as great as a facor of fifty, the time and cost graphs show a marked similarity of profile. Though the magnitudes are different, the relative position of the bars compared within instantiations remains largely the same.

While no one strategy is an across-the-board winner, QWU has the best time in four of the cases, and the best cost in three. Certain runs of STE capture the top place in the other cases. The BAF strategy trails in almost every case regardless of the sequencing used, failing to beat unsequenced

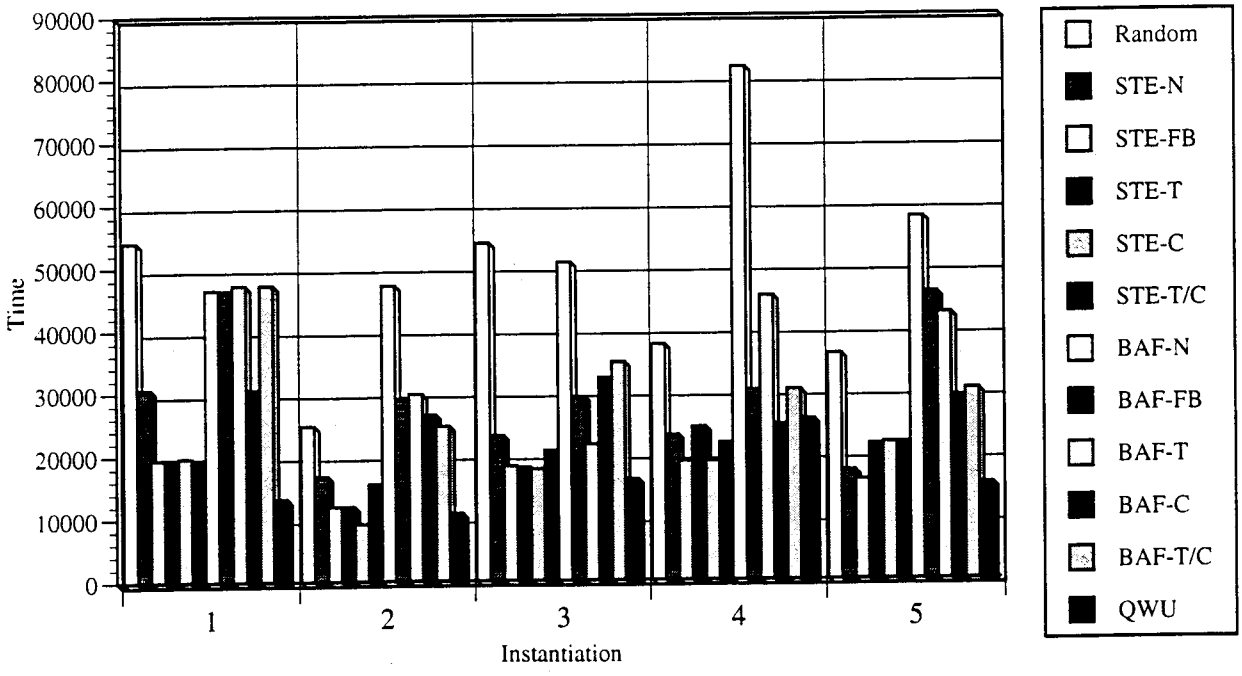

Figure 4. Time results for multiple instantiations of a ten module system.

STE in most cases. In three of the instantiations, some of the sequenced BAF trials fail to beat the random strategy in either time or cost. Note, however, that except for one instantiation, sequencing produces significant gains over unsequenced BAF.

From these trials, it is not possible to clearly pick a sequencing method which is best for either STE or BAF. In various cases, different sequences produce the best overall result for this strategy. For STE, sequencing for minimum feedbacks always produces a time and cost drop in the ten module system. This is consistent with previous results [5], however this does not hold true for the other systems.

In the interest of space, only time results are shown for the twenty and thirty module systems in figures 6 and 7 respectively. As with the ten module system, the cost results closely parallel these in relative magnitude with a few scattered anomalies. Additionally, since the BAF strategy shows similar poor results for the other systems, it is left off of these graphs in the interest of clarity.

Looking collectively at the QWU results across the systems, it is possible to see that, while this strategy may or may not produce the best achieved result, it is never far above it in cost or time.

Parallel Strategies

To demonstrate the parallel strategies, three systems, again of 10, 20 and 30 modules (though different from the above) were generated with the simulator. Each of these systems was then converged with both PB and SWU, using QWU as a reference. These results are presented in Table 1. Note from the values that for each of the systems, using a parallel strategy results in a marked drop in processing time, with parallel blitz showing the best time results. Just the opposite is true for cost, however, and it appears that the parallel strategies trade off cost for speed.

TABLE I. Results for perallel strategies.

System	Strategy	Time	Cost
10 module	QWU	10600	16949
	PB	2240	80954
	SWU	3237	15516
20 module	QWU	44191	47491
	PB	6604	321549
	SWU	14135	232931
30 module	QWU	128417	119176
	PB	13494	781635
	SWU	23141	431703

From observation of the running sequence, SWU rapidly deteriorates into PB as the queue becomes saturated. This is caused by the fact that completed modules queue multiple dependent modules faster than those modules can be cleared.

CONCLUSIONS

It is not possible to draw absolute across-the-board conclusions about a "best" strategy from these proof-of-concept runs because there are exceptions to every trend. Further and more complete exploration is needed to determine what techniques are universally best for complex system convergence. It is, however, possible to spot trends among this first generation of strategies. Sequencing does, in general, improve the performance of those strategies it affects. The BAF strategy may be ruled out as viable unless it receives some considerable alteration. The QWU strategy appears to be a "safe

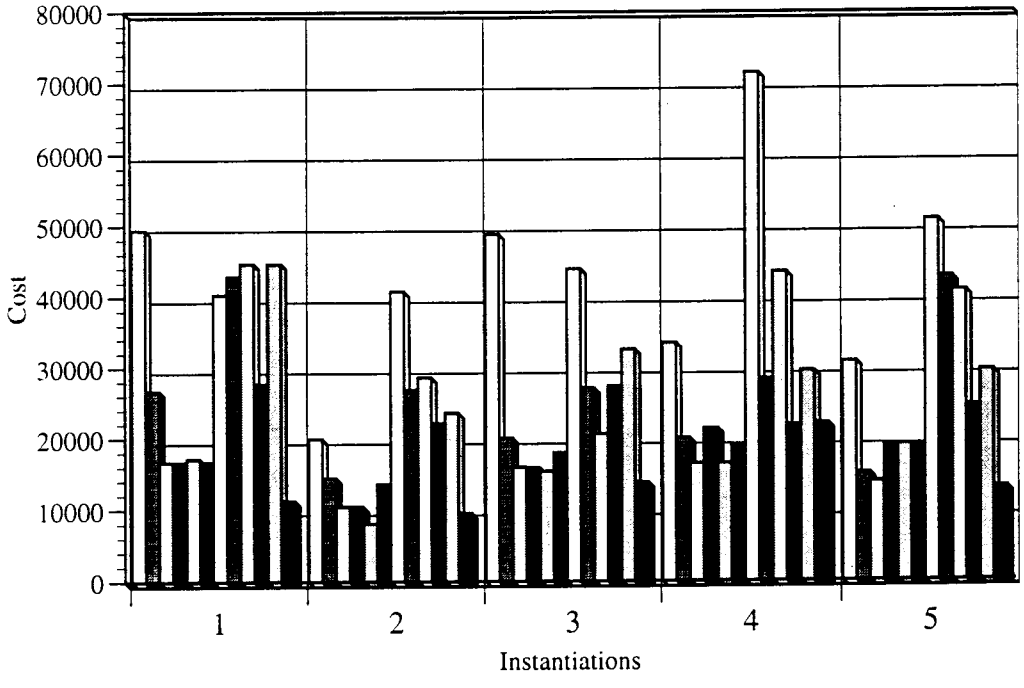

Figure 5. Cost results for multiple instantiations of a ten module system.

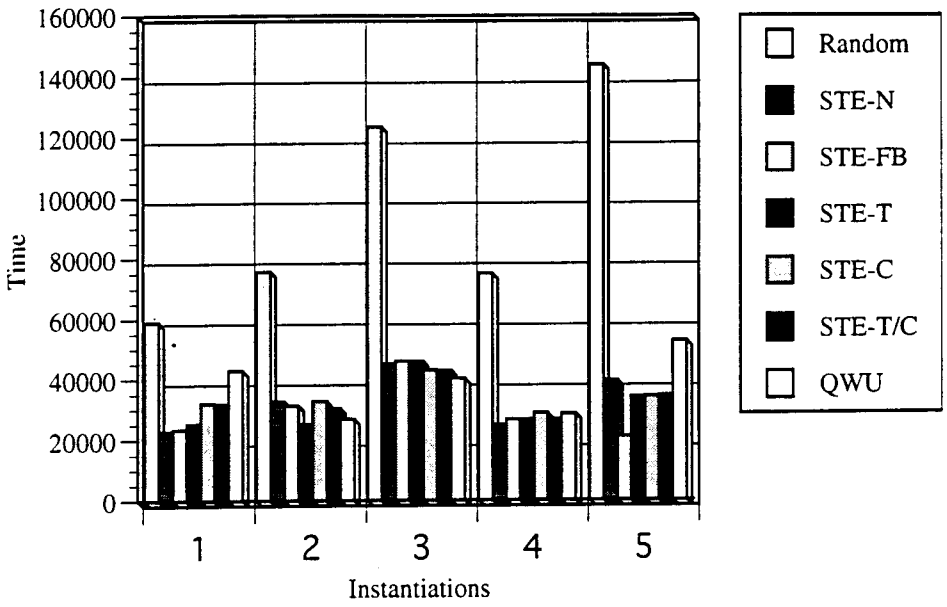

Figure 6. Time results for multiple instantiations of a 20 module system.

bet" choice among the sequential strategies explored since it consistently produces good, if not the best, results. Parallel strategies trade high cost for significantly reduced times.

Using simulation as a guiding technique, it will be possible to design and test more intelligent strategies. It should be possible with effort to maintain the time benefit of parallelism while driving down the cost to competitive levels. The use of coupling strengths derived from sensitivity information may enhance the convergence process. Concepts behind the influence of semantics on the system convergence, and possibly controls, may be discovered.

ACKNOWLEDGMENTS

Support for this work under NASA Langley Research Center grant NAG 11800 and NSF PFF grant DMI9553210 is gratefully acknowledged. The authors would also like to thank James L. Rogers of NASA Langley for his assistance.

REFERENCES

[1] Bloebaum, C. L.; (1995) Coupling Strength-Based System Reduction for Complex Engineering Design. *Structural Optimization*, Vol. 10, No. 2, Oct. 1995, p. 113-126.

[2] Goldberg, D.; (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Co., New York, 1989.

[3] Hajela, P.; Bloebaum, C. L.; Sobieski, J.; (1990) Application of Global Sensitivity Equations in Multidisciplinary Aircraft Synthesis. *Journal of Aircraft*, Vol. 27, No. 12, 1990, p. 1002-1010.

[4] Hulme, K. F.; Bloebaum, C. L.; (1996) Development of CASCADE: A Multidisciplinary Design Test Simulator. *Proceedings of the sixth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Seattle, WA, September 1996. To appear in *Structural Optimization*.

[5] McCulley, C.; Bloebaum, C. L.; (1996) Complex System Design Task Sequencing for Cost and Time Considerations. *Proc. of the Sixth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Seattle, WA, September 1996.

[6] McCulley, C.; Bloebaum, C. L.; (1996) A Genetic Tool for Optimal Design Sequencing in Complex Engineering Systems. *Structural Optimization*, Vol. 12, No. 2/3, Oct. 1996, p.186-201.

[7] McCulley, C.; Bloebaum, C. L.; (1994) Optimal Sequencing for Complex Engineering Systems Using Genetic Algorithms. *Proc. of the Fifth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Panama City, FL, September 1994.

[8] Rogers, J. L.; Bloebaum, C. L.; (1994) Ordering Design Tasks Based on Coupling Strengths. *Proc. of the Fifth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Panama City, FL, September 1994.

[9] Rogers, J. L.; McCulley, C.; Bloebaum, C. L.; (1996) Integrating a Genetic Algorithm Into a Knowledge-Based System for Ordering Complex Design Processes. *Proceedings of the International Conference on AI in Design*, Stanford, CA, June 1996.

[10] Rogers, J.L.; (1989) A Knowledge-Based Tool for Multilevel Decomposition of a Complex Design Problem, NASA TP 2903.

[11] Shewchuk, J.P.; Chang, T.C.; (1991) An approach to object-oriented discrete event simulation of manufacturing Systems. Proceedings Winter Simulation Conference, Phoenix, AZ, 1991.

[12] Sobieski, J.; (1993) Multidisciplinary Design Optimization: An Emerging New Engineering Discipline. NASA Technical Memorandum 107761.

[13] Sobieski, J.; (1990) Sensitivity of Complex, Internally Coupled Systems. *AIAA Journal*, Vol. 28, No. 1.

[14] Starkweather, T.; McDaniel, S.; Mathias, K.; Whitley, D.; Whitley, C.; (1991) A Comparison of Genetic Sequencing Operators. *Proc. of the Fourth International Conference on Genetic Algorithms*.. Proceedings published by Morgan Kaufmann.

[15] Steward, D.V.; (1981)*System Analysis and Management*. Petrocelli Books, New York, 1981.

[16] Syswerda, G.; (1990) Schedule Optimization Using Genetic Algorithms. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1990.

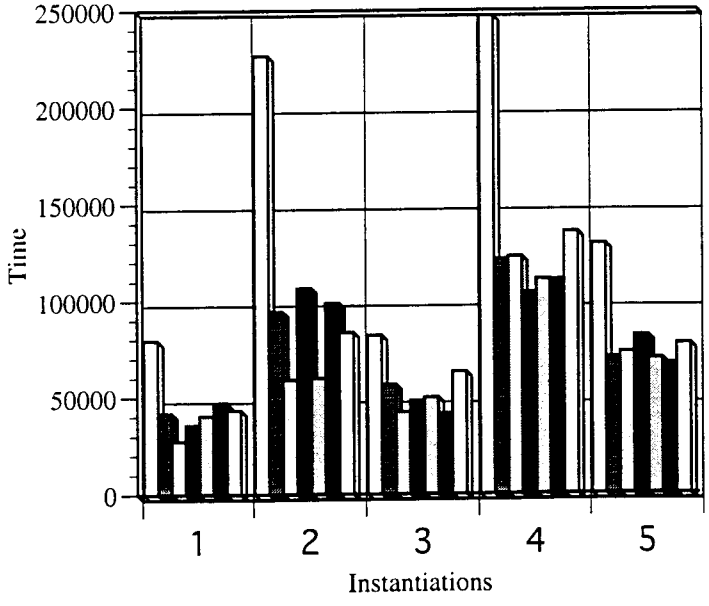

Figure 7. Time results for multiple instantiations of a 30 module system.