JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, September 2006

Aerospace Vehicle Trajectory Design and Optimization Within a Multi-Disciplinary Environment

Robert Windhorst*
NASA Ames Research Center, Moffett Field, CA 94035
and
Eric Galloway, Eric Lau,† David Saunders,‡ and Peter Gage§
ELORET, Moffett Field, CA 94035

A trajectory design and optimization tool named *Mission* is developed to operate within a multi-disciplinary analysis tool environment for conceptual design of aerospace vehicles. *Mission* possesses several features designed to facilitate its set-up and operation within the environment. It receives input via an Extensible Markup Language tagged data file. The tree structure of the tags reflects that of a branched and multi-segmented trajectory, aiding parsing and editing for set-up of data exchange within the environment. *Mission* uses Xerces, a public domain library, for parsing data files. In addition, it is linked to both gradient-based and genetic algorithm-based optimizers, allowing a choice depending on the amount of *a priori* information available. When little is known about the control variables, an initial guess of their histories may be forgone by initializing the optimization process with the genetic algorithm. Once a solution is approached, the gradient-based method is used. This strategy increases the robustness and autonomy of the trajectory tool operation within the multi-disciplinary environment. Finally, *Mission*'s trajectory integration process is coded for parallel execution via the Message Passing Interface standard. The resultant execution speed increase reduces the relative expense of operating *Mission* within the multi-disciplinary environment. The results of a Crew Transfer Vehicle "abort from ascent" problem are presented to demonstrate and quantify *Mission*'s features.

Nomenclature

a	base area of nozzle
c	fuel mass flow rate
C_D	drag coefficient
C_L	lift coefficient
D	drag
g	local gravitational acceleration
g_o	sea level gravitational acceleration
h	altitude
I_{sp}	specific impulse
L	lift
m	mass

Received 27 January 2005; accepted for publication 5 October 2005.

* Aerospace Engineer, Senior Member AIAA.
† Research Scientist.
‡ Senior Research Scientist, Member AIAA.
§ Technical Director, Associate Fellow AIAA.

n_{dv}	number of design variables
n_g	number of trajectory integrations needed to calculate a gradient
n_{gen}	number of generations per execution
n_l	number of trajectory integrations needed to perform a line search
n_i	number of major iterations per execution
n_p	number of processors
n_{pop}	number of genetic strings in a population
M	Mach number
p	local atmospheric pressure
p_c	probability of a gene being crossed over
p_m	probability of a gene being mutated
p_{sl}	sea level atmospheric pressure
q	dynamic pressure
r	radial coordinate
r_e	radius of earth
r_s	execution speed ratio of two parallel runs
S	side force
S_{ref}	vehicle reference area
t	time
t_i	time required to perform a gradient-based optimization iteration
t_g	time required to evaluate fitness function values for all genetic strings in population
t_s	average time required to integrate a single trajectory
T	thrust
v	relative speed of vehicle
α	angle of attack
β	side slip angle
χ	heading angle
δ	nozzle pitch angle
ϕ	longitude
γ	flight path angle
η_{feas}	SNOPT feasibility criterion
η_{opt}	SNOPT optimality criterion
λ	latitude
μ	throttle setting
ρ	local atmospheric density
σ	bank angle
ω	angular speed of earth
$\mathbf{A}$	a matrix
$\mathbf{l}$	vector of lower bounds on $\mathbf{x}$
$\mathbf{u}$	vector of upper bounds on $\mathbf{x}$
$\mathbf{x}$	vector of design variables
$f(\mathbf{x})$	objective function
$\mathbf{c}(\mathbf{x})$	vector of nonlinear constraint functions

I. Introduction

A NEW trajectory design and optimization tool called *Mission* is being developed at NASA Ames Research Center. It is intended to work within the Advanced Engineering Environment (AEE), a multi-disciplinary analysis tool environment for conceptual design of future launch vehicles linked via a Product Data Management (PDM) system to a relational database.[1]

As an integral part of the vehicle conceptual design process, the trajectory code calculates the fuel burned during ascent for vehicle sizing and the heat load encountered during re-entry for Thermal Protection System (TPS) material choice and thickness sizing.[1,2] Fuel burned or heat load is desired not for just any feasible ascent or re-entry trajectory, but rather for an optimal trajectory–that is, a trajectory that minimizes fuel consumed, heat load, or any other parameter of interest. The trajectory code must therefore have the ability to integrate many trajectories and identify optimal control histories. In the case of an abort, the trajectory code serves to identify viable trajectories given the current state of the vehicle as needed for safety probability determination.[1,3] Furthermore, the trajectory code must perform these analyses for many different vehicle concepts and for trajectories defined by different origins, destinations, and events.

The POST[4] and OTIS[5] trajectory tools, currently available within AEE, are the products of many years of development. Both have the capability of modeling most vehicles and trajectories and are based on tested and sound algorithms. Unfortunately, their heritage code structure makes them unwieldy to set up within AEE. For example, they use long detailed FORTRAN namelists for data input. Namelists exhibit minimal data structure and do not support the inclusion of data attributes, making them difficult to parse for information. Moreover, POST and OTIS are linked to gradient-based optimizers that generally require good initial guesses of the design variables to produce sensible answers. This burdens the user and hinders autonomous execution when linked with other tools.[1] Depending on the initial starting point, POST and OTIS optimized solutions can easily require hours of execution time.[1]

A newer trajectory tool, *Traj_Opt*,[3] has also been integrated into the AEE environment. *Traj_Opt* employs a gradient-based optimizer and, while effective and efficient on a single processor, shares lack of autonomy with POST and OTIS. In addition, *Traj_Opt* treats single segment trajectories only and has no propulsion capabilities.

Experience with the several existing trajectory tools available in AEE prompted Monell et al., in Ref. 1 to note that "trajectory calculations consume a significant portion of the total analysis cycle for two reasons: optimization is inherently computationally expensive . . ., and many optimizations fail, requiring user intervention." *Mission* offers two main features designed to reduce the relative expense of trajectory calculations within AEE: optimizer interchangeability and parallelization.

In addition to the features listed above, *Mission*'s input file contains data marked up with Extensible Markup Language (XML). The tree structure of the XML tags reflects that of branched and multi-segmented trajectories, thereby facilitating data parsing. The relative ease of data parsing reduces the complexity of setting up data exchange between *Mission* and other tools in the environment.

Mission's code is object-oriented and written in C++. This makes its various algorithms and functions independent of each other.[6] Hence, if revisions or extensions are desired, the changes to the code are localized. Moreover, unit testing the algorithms and functions is facilitated.

This paper presents the trajectory integration and optimization algorithms within *Mission*. It shows how the tree structure of the XML tags reflects that of a branched and multi-segmented trajectory, and it describes the strategy used to parallelize the trajectory integration portion of the optimization process. Finally, the results of an abort from ascent optimization problem are presented to demonstrate and quantify *Mission*'s features.

II. Trajectory Integration

Mission generates trajectories by integrating the three-degrees-of-freedom equations of motion for a point mass vehicle traveling in the vicinity of a spherical earth with no winds.[7] Six flight path coordinates are used as state variables. There are seven equations, one for each state variable:

$$\dot{\phi} = \frac{v}{r\cos\lambda}\cos\gamma\cos\chi \tag{1}$$

$$\dot{\lambda} = \frac{v}{r}\cos\gamma\sin\chi \tag{2}$$

$$\dot{r} = v\sin\gamma \tag{3}$$

$$\dot{v} = \frac{1}{m}(T\cos\beta\cos(\alpha+\delta) - D) - g\sin\gamma + r\omega^2\cos\lambda(\sin\gamma\cos\lambda - \sin\lambda\sin\chi\cos\gamma) \tag{4}$$

$$\dot{\chi} = \frac{1}{mv\cos\gamma}[\cos\sigma(S + T\sin\beta\cos(\alpha+\delta)) - \sin\sigma(L + T\sin(\alpha+\delta))] - \frac{v}{r}\cos\chi\cos\gamma\tan\lambda + 2\omega(\sin\chi\cos\lambda\tan\gamma - \sin\lambda) - \frac{r\omega^2}{v\cos\gamma}\sin\lambda\cos\lambda\cos\chi \tag{5}$$

$$\dot{\gamma} = \frac{1}{mv}[\cos\sigma(L + T\sin(\alpha+\delta)) + \sin\sigma(S + T\sin\beta\cos(\alpha+\delta))] + \cos\gamma\left(\frac{v}{r} - \frac{g}{v}\right) + 2\omega\cos\lambda\cos\chi + \frac{r\omega^2}{v}(\cos\gamma\cos^2\lambda + \sin\lambda\sin\chi\sin\gamma\cos\lambda) \tag{6}$$

and

$$\dot{m} = c. \tag{7}$$

Mission uses several mathematical models for calculating forces acting on the vehicle. The 1976 Atmosphere Model[8] supplies local atmospheric properties about the vehicle as a function of altitude. Vehicle aerodynamic coefficients, C_L and C_D, are calculated via tri-linear interpolation of pre-computed three dimensional tables. The table independent variables are M, q, and α. L and D are then calculated via

$$L = \frac{1}{2}\rho v^2 S_{ref} C_L \quad \text{and} \quad D = \frac{1}{2}\rho v^2 S_{ref} C_D. \tag{8}$$

The local acceleration of gravity is modeled by a simple inverse square law,

$$g = g_o\left(\frac{r_e}{h + r_e}\right)^2 \tag{9}$$

Thrust is modeled by the relationship

$$T = \mu c g_o I_{sp} - a(p_{sl} - p) \tag{10}$$

The object-oriented C++ code structure of *Mission* allows convenient incorporation of new math models to extend or replace those described above.

Figure 1 shows a block diagram of the *Mission* code. There are two types of inputs to the trajectory integration algorithm: boundary conditions and controls.

Boundary conditions are also of two types: initial and final. Initial conditions denote the starting point of the trajectory. Fully specified initial conditions consist of values for each of the seven state variables and a beginning time. A final condition denotes an event occurring in the middle of, or at the end of a trajectory. For example, a first stage separation is an event that could occur in the middle of a trajectory. A fully specified final condition consists of a single value of one of the seven states, time, or a function of these. Currently, *Mission* possesses a library of functions that may be used as boundary conditions. *Mission*'s object-oriented code structure allows convenient extension of this library.

Four controls are available to the user: α, σ, μ, and δ. Each control must be specified as a function of time. Within *Mission*, functions of time are modeled using spline interpolations on a user-supplied one-dimensional table with t as the independent variable. *Mission*'s object-oriented code structure allows convenient extension of the set of control variables and swapping of t with a state variable.

The outputs of the trajectory integration algorithm are the time histories of the states, state rates, controls, and functions of the states. Generally, functions of the states include local atmospheric parameters, heating parameters

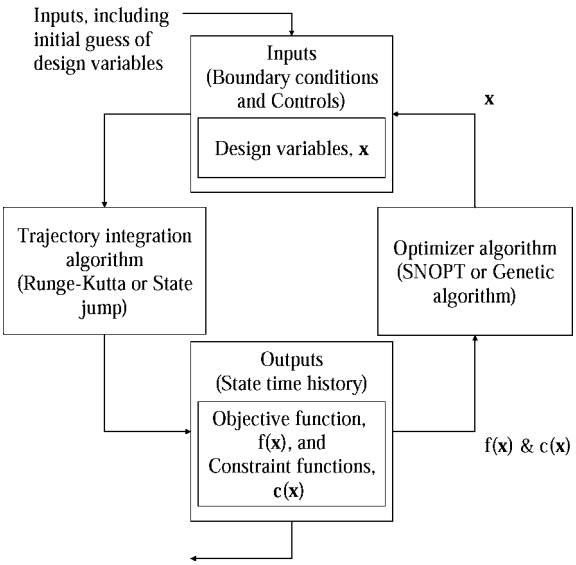

Fig. 1 *Mission* block diagram.

for specified locations on the vehicle body, and forces acting on the body. *Mission* possesses a library of functions that can be used for output. As with the boundary conditions, *Mission*'s object-oriented code structure allows convenient extension of this library.

In addition to its $4^{th}/5^{th}$ order Runge-Kutta integration algorithm, *Mission* also contains a state jump algorithm for modeling trajectory segments such as rocket staging. As with the math models, incorporation of other trajectory integration algorithms is facilitated by *Mission*'s object-oriented code structure.

III. Trajectory Optimization Method

Mission identifies optimal trajectories via the Direct Shooting[9] method. The general nonlinear constrained optimization problem may be stated mathematically as[10]

$$\min_{\mathbf{x}\in\Re^n} f(\mathbf{x})$$

$$\text{subject to } \mathbf{l} \le \begin{pmatrix} \mathbf{x} \\ \mathbf{c}(\mathbf{x}) \\ \mathbf{A}\mathbf{x} \end{pmatrix} \le \mathbf{u}. \tag{11}$$

Trajectory optimization by Direct Shooting is achieved by choosing a subset of the inputs to the trajectory integration algorithm as design variables, see Fig. 1. Thus, design variables may be values of boundary conditions and/or controls. When a control is chosen as part of the set of design variables, the values of the dependent variables in the table that model the control are added to the design variable set.

As illustrated in Fig. 1, a subset of the outputs of the trajectory integration algorithm is specified as objective and constraint function values. Therefore, state, state rate, control, and/or functions of state values at any event time or all times during a trajectory segment may be used as an objective or constraint function.

The user initializes the entire process by supplying values for all of the inputs. Since the design variables are part of the inputs, the user normally supplies an initial guess of their optimal values.

Mission provides the user with two optimization methods, a gradient-based method and a genetic algorithm(GA)-based method. The gradient-based method is SNOPT, a sequential quadratic programming method fully described in Ref. 10. The GA-based optimizer was written at Ames Research Center and is further described below. As with the trajectory integration algorithms, incorporation of other optimization algorithms is facilitated by the object-oriented code structure.

IV. Genetic Algorithm

Gradient-based optimizers applied to trajectory optimization are very efficient given a good guess of the design variables and accurate derivatives of the objective and constraint functions. They do not, however, guarantee convergence on a *global* optimum from any given specific guess. Further problems can occur when a good guess is not known. From a poor guess, a locally optimum solution is frequently unacceptable to the user. More commonly unconverged results may be produced with some constraints apparently infeasible. For many problems, the user may have to iterate by supplying the optimizer with multiple guesses until the optimizer returns an acceptable solution. This process requires considerable user intervention and experience.[1]

In contrast to gradient-based approaches, GA-based optimizers generate their own guess and are robust in the face of discontinuous or nonlinear functions. Their disadvantage is that in practice they are costly in the sense that they require large numbers of calls to the objective function for relatively small decreases in the cost function.[11] Treatment of constraints is also limited.

The GA linked to *Mission* is a simple one. It was created to explore the possibility of using a GA to autonomously generate a guess of the design variable values that could be used to initialize a gradient-based optimization algorithm. A selection strategy (described below) possessing weak pressure was used to avoid quick convergence to a local optimum. More sophisticated selection strategies with stronger pressure appear in the literature.[12]

Mission's GA uses several steps. First, a population of genetic strings, each containing a full set of design variable values, is generated randomly. The design variable values in the genetic string must be between their upper and lower limits. This population is the zeroth generation which is subsequently morphed so that each member of its successive generations improves the value of a fitness function. The fitness function here is the objective function of the optimization problem. If the optimization problem is constrained, a weighted sum of the squares of the constraint violations is added to the fitness function. The fitness function value of each genetic string in the population is evaluated by using the design variable values stored in the genetic string to integrate a new trajectory and compute objective and constraint values. Fitness function values are stored with their corresponding genetic strings.

New generations are formed by the following selection process. The population is randomly grouped in pairs of genetic strings. Each pair is taken from the population as parents and used to spawn two children through a combination of mutation and cross-over processes. Fitness function values are evaluated for each child. Of the two parents and two children, the two genetic strings with the best fitness are placed back into the population, while the others are discarded.

While the robustness of the GA-based optimizer is desired in regions far away from an acceptable solution, the convergence property of the gradient-based optimizer is desired when near an acceptable solution. *Mission* easily interfaces with either type of optimizer. Hence when confronted with a problem without a good starting point, it can use the GA-based optimizer until a suitable guess is determined and then switch to a gradient-based algorithm to refine the solution. In practice, the switch is made when the GA-based optimizer is no longer making acceptable progress. Also, in the example problem that we solved we increased the size and density of the set of control variables when making the switch. Within AEE this process could be automated such that no user intervention was required.

V. Input File Data "Marked Up" with XML

Mission's input file data is "marked up" using XML, the W3C-endorsed standard for document mark up.[13] Within XML a unit of data and its associated mark up tags are called an element (see Fig. 2). Mark up tags delimit the element and contain the element name and attributes. Both beginning and ending tags are surrounded by angle brackets, with the ending tag containing a slash after the first bracket. Elements contain data, in the form of text strings and/or other elements. An XML schema defines the structure of the elements, i.e., names, contents, and attributes. Schemas can be developed for any set of data a developer wishes to structure and store in a computer.

XML is fast becoming an industry standard for marking up data passed between servers and web browsers.[13] In fact there already exists a large amount of support in the form of free tools for defining, parsing, and editing XML. In addition, many database servers provide tools designed to support XML. Routine operations, such as parsing files

<elementName attributeName=attributeValue>elementData</elementName>

Fig. 2 Structure of an XML element.

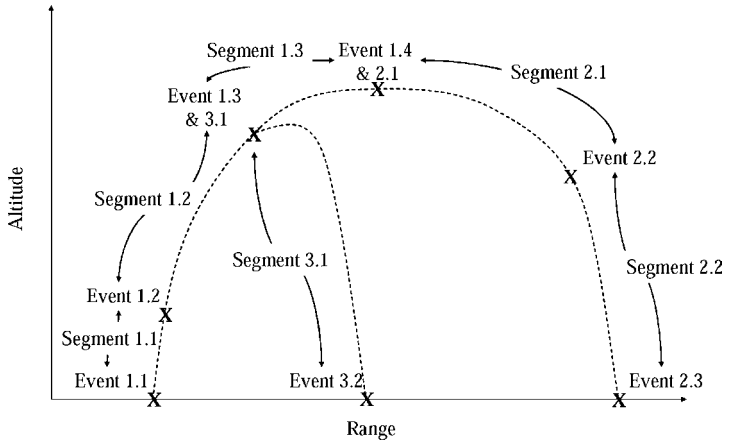

Fig. 3 Generic RLV mission.

and passing data to other processes such as the PDM, are naturally supported by the environment instead of the programmer.

Mission is linked with the Apache implementation of the Document Object Model (DOM) named Xerces[14] for internal reading and parsing of the input file data. *Mission*'s data elements and element contents are defined by an XML schema. When *Mission* is executed, Xerces automatically validates the input file with the schema to find syntax and context errors. When Xerces encounters an error, it provides a clear message that assists the user with quickly identifying the error in the input file and correcting it.

A data model named the Launch Vehicle Language (LVL) was developed using XML to ensure consistent data passage between tools within AEE.[15,16] Because both *Mission* input data and the LVL are implemented in XML, *Mission* input data essentially becomes an extension of the LVL. The weights and sizing tool, XWAT, also has adopted XML for the same reason.[17]

The XML schema defines element tags and content such that they model the tree structure of the mission being analyzed. Figure 3 and Table 1 illustrate the structure of a generic reusable launch vehicle (RLV) mission with the on-orbit section removed. Notice that it consists of components made up of other components forming a tree. For example, a mission component is on the first level and consists of a group of trajectory components each bounded by event components. Similarly, on the second level, trajectory components consist of a group of segment components each bounded by event components. Mission, trajectory, event, and segment XML elements may be defined accordingly.

Figure 4 shows one method of marking up with XML the data that describe the generic RLV mission presented in Fig. 3 and Table 1. The elements are missions, trajectories, segments, and events. Each element has a name attribute

Table 1 Generic RLV mission trajectories, segments, and events.

Trajectory	Segment	Event
1. Ascent	1.1 Clear tower	1.1 Ignition
	1.2 1st Stage	1.2 Tower clearance
	1.3 2nd Stage	1.3 Stage separation
		1.4 Orbit insertion
2. Re-entry	2.1 Re-entry	2.1 De-orbit
	2.2 Max L/D glide	2.2 Max heating
		2.3 Landing
3. Tank return	3.1 Tank return	3.1 Stage separation
		3.2 Splash down

```
<mission name=RLV Generic Mission>
    <trajectory name=Ascent>1
        <event name=Ignition>1.1</event>
        <segment name=Clear tower>1.1</segment>
        <event name=Tower clearance>1.2</event>
        <segment name=1st Stage>1.2</segment>
        <event name=Stage separation>1.3</event>
        <segment name=2nd Stage>1.3</segment>
        <event name=Orbit insertion>1.4</event>
    </trajectory>
    <trajectory name=Re-entry>2
        <event name=De-orbit>2.1</event>
        <segment name=Re-entry>2.1</segment>
        <event name=Max heating>2.2</event>
        <segment name=Max L/D glide>2.2 </segment>
        <event name=Landing>2.3</event>
    </trajectory>
    <trajectory name=Tank return>3
        <event name=Stage separation>3.1</event>
        <segment name=Tank return>3.1</segment>
        <event name=Splash down>3.2</event>
    </trajectory>
</mission>
```

Fig. 4 XML Mark up of data describing an RLV generic mission.

corresponding to its name in Table 1 and data corresponding to its number in Table 1. *Mission's* XML mark up structure is similar to that shown in Fig. 4. It also contains the same mission, trajectory, segment, and event elements. However, their data and attributes are different and more extensive, designed to contain all the information necessary to set up the program. In addition, it possesses many more types of elements. A wide range of missions can be created by arrangement of the elements. Once missions are created, the readable format of the data facilitates modification. If something more sophisticated than a text editor is desired to do the modification, free or commercial XML editing tools are available.[16]

Not only is the tree structure of the mission reflected by the XML, but it is also reflected by the class structure of the object-oriented code. For example, the trajectory class definition contains pointers to lists of segment and event objects. Reuse of the mission tree structure at multiple development stages of the code (mission design, input file data mark up, and object-oriented design) minimizes the need to restate, reformat, or translate data, when using, modifying, or testing the code.

VI. Parallelization

The trajectory integration portion of the optimization process is parallelized for increased execution speed. The Message Passing Interface[18] (MPI) standard is employed for portability. The parallelization scheme is slightly different depending on the optimization algorithm being used. Relations for estimating the speed ratio of two runs utilizing different numbers of processors for each optimization algorithm are derived and discussed below.

The process of evaluating fitness functions in the GA is conveniently parallel because the fitness function value for each genetic string can be evaluated independently. Each fitness function evaluation consists of a trajectory integration and an evaluation of the output functions. Hence, the time required to calculate all the fitness function values of a population of genetic strings is estimated as

$$t_g \approx n_{pop} t_s. \tag{12}$$

Here, t_s denotes the average value of the time required to integrate a single trajectory. In practice this time varies significantly depending on trajectory length and time step. The n_{pop} fitness function evaluations can be distributed across n_p processors. Now,

$$t_g \approx r_{up} \left[\frac{n_{pop}}{n_p} \right] t_s. \tag{13}$$

where r_{up} denotes a round up function. A round up function rounds up all numbers having a decimal portion, no matter how small. Finally the execution speed ratio of two runs possessing the same number of generations is

$$r_s \approx \frac{r_{up}\left[n_{pop_1}/n_{p_1}\right]}{r_{up}\left[n_{pop_2}/n_{p_2}\right]} \tag{14}$$

where the subscript denotes run 1 or run 2.

Gradient-based optimization algorithms compute optimal design variable values by successively improving on an initial guess. A single iteration consists of two parts, 1) finding a search direction and 2) performing a line search along the direction found in 1). The line search identifies new values of the design variables that produce more optimal values of the objective and constraint functions. Our parallelization effort focuses on the former part.

An integral part of finding a search direction is calculating the gradients of the objective and constraint functions. In our case, gradients are generated by finite differencing each design variable and integrating a new trajectory to obtain the differences in objective and constraint function values. This process is simple to parallelize because, like the process for evaluating fitness functions of genetic strings, each trajectory can be calculated independently.

The time required to perform a single iteration is estimated as

$$t_i \approx t_s(n_g + n_l). \tag{15}$$

t_s is essentially constant for the finite difference calculations because the control finite differences are necessarily tiny. However, t_s can vary significantly from one iteration to the next.

Although only one gradient calculation per iteration is assumed here, sometimes SNOPT requests more than one. Equation (15) assumes that the time required for trajectory integrations is much greater than the time required for other operations during the iterations. Now, assuming one-sided differences,

$$n_g = n_{dv} + 1, \tag{16}$$

where the addition of one accounts for the nominal trajectory. For central differences this would become

$$n_g = 2n_{dv} + 1. \tag{17}$$

As with the distribution of processors for the GA method, the n_{dv} integrations are distributed across n_p processors. Note that the nominal trajectory is not included in the distribution. Now,

$$t_i \approx t_s(r_{up}\left[\frac{n_{dv}}{n_p}\right] + 1 + n_l), \tag{18}$$

Finally, the speed ratio of two runs having the same number of optimization iterations is

$$r_s \approx \frac{r_{up}[n_{dv_1}/n_{p_1}] + 1 + n_{l_1}}{r_{up}[n_{dv_2}/n_{p_2}] + 1 + n_{l_2}} \tag{19}$$

Equations 14 and 19 ignore the additional set-up time for multiple processors and the time required for the multiple processors to communicate with each other. In practice these become significant as n_{dv} approaches n_p, as will be shown below.

VII. HL-20 Abort to Gander

Mission was applied to an "abort from ascent" trajectory optimization problem to demonstrate its optimization and execution speed features. This is the same problem as one of those worked in Ref. 3. In this scenario a Crew Transfer Vehicle (CTV) conceived of in Ref. 19, the HL-20, is carried atop a Titan III expendable launch vehicle (ELV). Launch occurs at the Kennedy Space Center in Florida. At some time during the ascent to orbit an abort is initiated and separation occurs. Of interest is the window of opportunity on the ascent trajectory from which the HL-20 can glide to Gander, Newfoundland. The edges of the window are identified by setting up a mission and applying two

optimizations to it. The mission consists of two trajectories: ascent and glide back. The ascent trajectory is fixed by the nominal mission. The glide back trajectory emanates from the ascent at separation and is modifiable. The lower edge of the window is found by minimizing the separation time subject to several constraints on the glide back trajectory: 1) the HL-20 must be able to glide to Gander, 2) the HL-20 does not exceed its temperature limit at the nose stagnation point, and 3) the HL-20 does not exceed a 3 g acceleration limit. In this case, the temperature limit is enforced via an Aero-thermal Performance Constraint (APC) curve.[20] This curve is pre-defined by coordinate pairs of speed and altitude such that, as long as the vehicle's speed and altitude place it above the APC curve, the temperature limit is not violated. Mission also supports distributed heating constraints via table look-up, but the APC alternative for a single surface location is convenient here for illustrative purposes. The upper edge of the window is found by maximizing the separation time subject to the same constraints. The design variables are the α and σ control points during the glide back along with the separation time which also serves as the objective.

The following discussion focuses on the results of the *latest* separation time optimization problem. Optimization was accomplished by first using the GA-based optimizer to generate an initial guess of the values of a coarse set of design variables. Once a suitable guess was generated, a denser set of design variables was swapped in, and SNOPT was used to refine the guess. Table 2 lists parameters of the GA and SNOPT optimization runs. These runs were made on a single processor 1,700 MHz Pentium 4 PC.

In the following figures, two curves are shown: the GA solution and the SNOPT solution.

Figure 5 illustrates the latitude vs. longitude path of the entire mission. The ascent trajectory of the stack-up is denoted by the squares, and the glide back trajectories of the HL-20 are denoted by the dotted lines. The separation point defines where the HL-20 separates from the stack-up, the glide back trajectory begins, and the top edge of the abort window. The GA and SNOPT solutions have slightly different separation times. The SNOPT separation time is later, thus more optimal, than the GA solution. The location of Gander is marked with a dot. Although the GA solution does not reach Gander exactly, it comes close enough that it provides an adequate initial guess for SNOPT.

Figure 6 shows the altitude vs. speed path of the glide back trajectory and the APC curve. Clearly, both glide back solutions remain above the APC curve, satisfying the temperature constraint.

Figure 7 shows the HL-20 angle of attack and bank angle time histories for the glide back trajectories. The SNOPT solution control angles are more refined, partly through use of more control points and partly through precise meeting of the necessary conditions for optimality. Note that employing comparable numbers of control variables for the GA tends *not* to refine the starting guess for SNOPT, apart from being too costly.

Figure 8 shows the HL-20 acceleration time histories for the glide back trajectories. Clearly, the 3 g acceleration limit is not violated by the SNOPT solution, but it is somewhat violated by the GA solution. The violation of the GA solution does not destabilize the SNOPT convergence.

As shown by the figures, the GA solution does not meet all the constraints and does not reach as optimal a solution as the SNOPT solution. By changing a constraint's weight in the fitness function, more or less emphasis can be placed on satisfying that constraint. GAs do not have defined ending criteria. Generations are spawned until no further improvement in the fitness function is observed. In our case we did not produce an exhaustive number of

Table 2 Parameters of genetic algorithm and SNOPT optimization runs

Genetic algorithm		SNOPT	
n_{dv}	17	n_{dv}	81
# αs	8	# αs	40
# σs	8	# σs	40
wall t	3,080 sec.	wall t	956 sec.
separation t	424 sec.	separation t	430 sec.
n_{gen}	50	n_i	104
n_{pop}	50	n_l	7.6
p_c	0.8		
p_m	0.1		

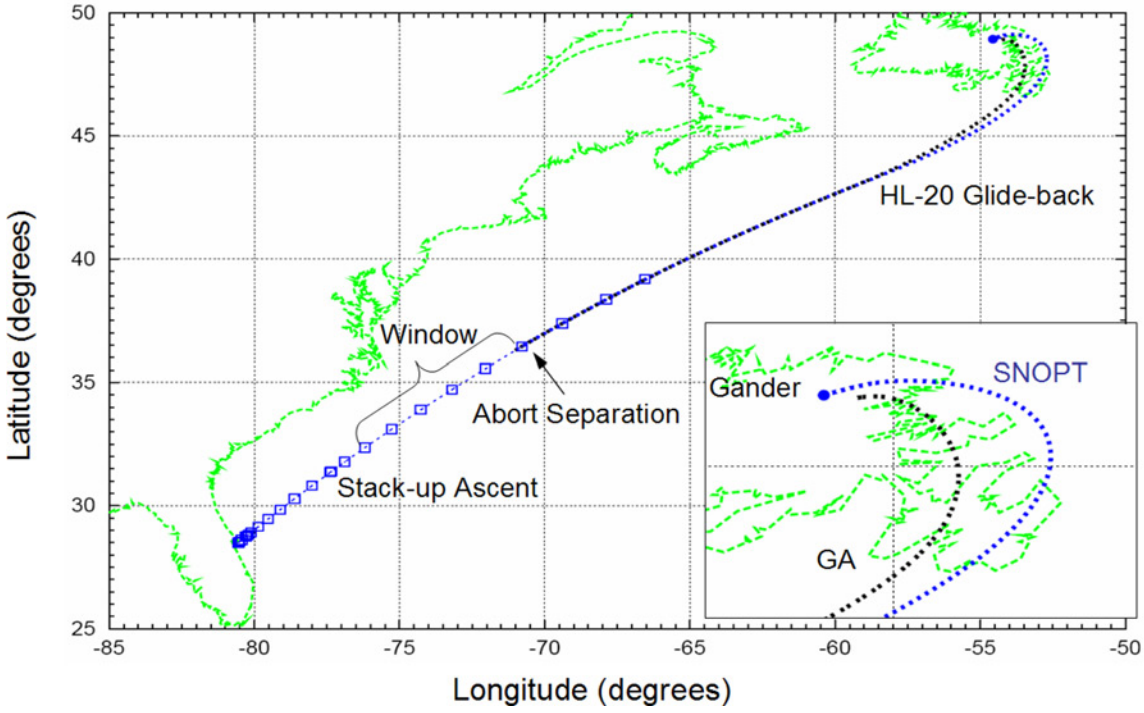

Fig. 5 Latitude vs. Longitude path for the entire mission.

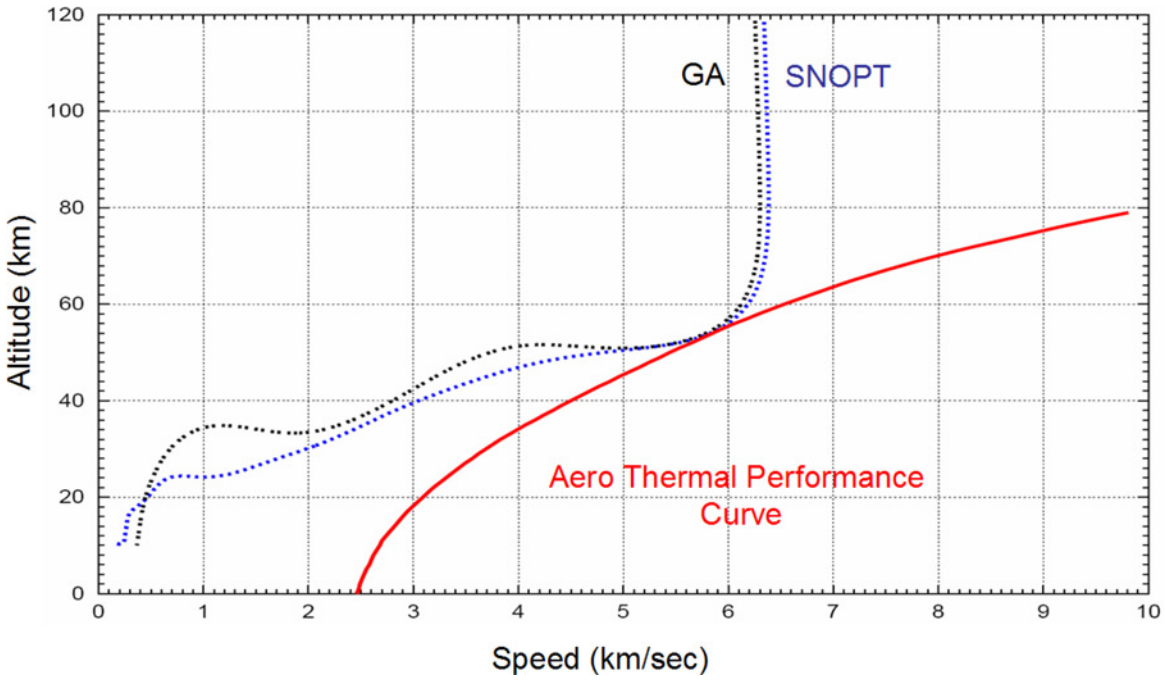

Fig. 6 HL-20 Altitude vs. speed path for the glide back trajectory.

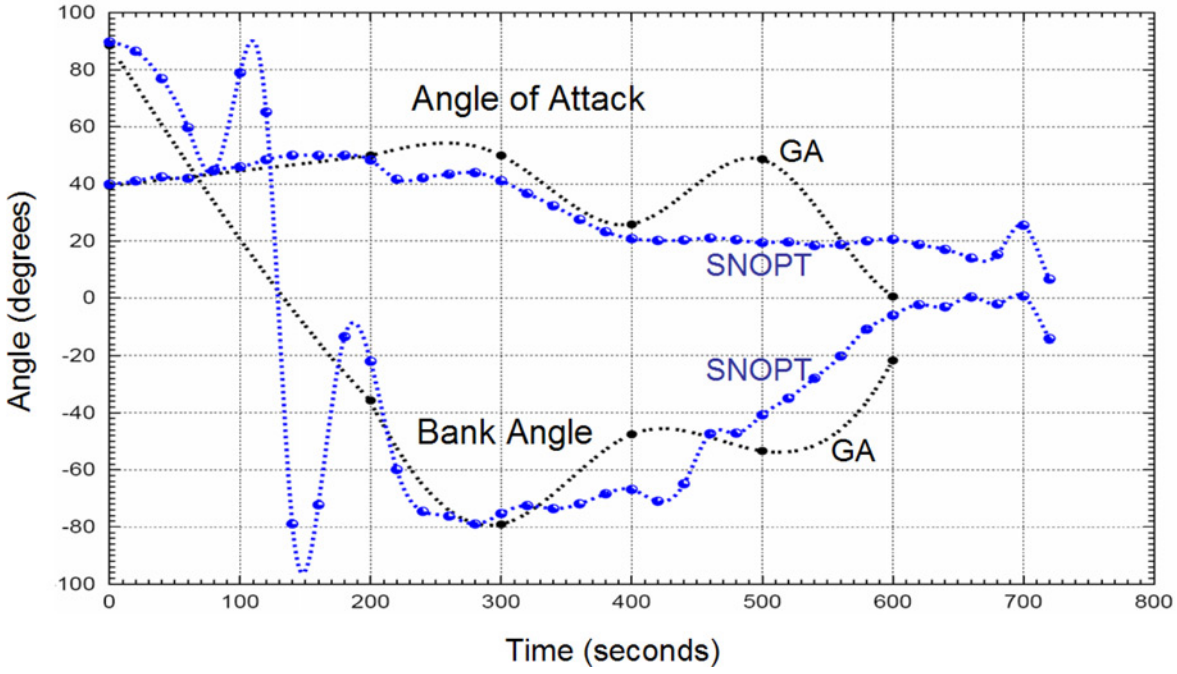

Fig. 7 HL-20 control angles for the glide back trajectory.

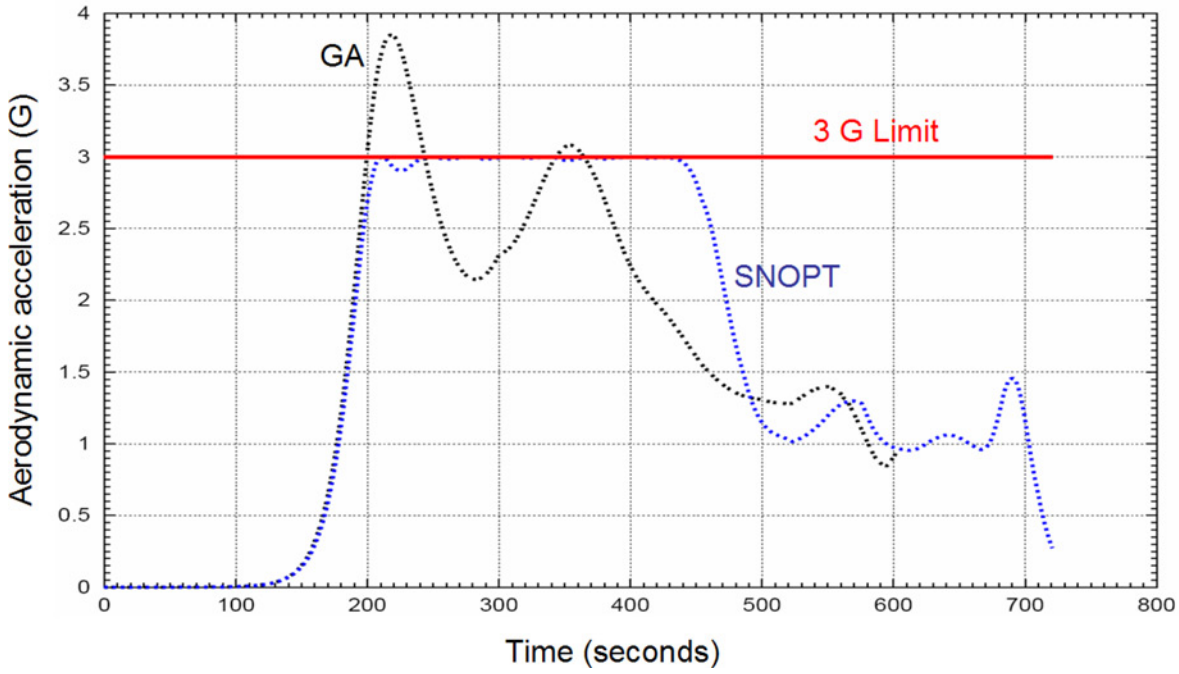

Fig. 8 HL-20 Aerodynamic acceleration for the glide back trajectory.

Table 3 NAS Machine specifications.

Name	Make	CPU	# CPUs	Clock
Lomax	SGI O3K	R12000	512	450 MHz

generations. We only produced enough to give us an adequate initial guess. Further generations may have produced solutions having a more optimal separation time and satisfying all of the constraints.

To illustrate the execution speed increase due to parallelization, multiple runs were made using different numbers of processors. For these runs, only the gradient-based optimizer was used. Similar execution speed increases are implied for runs made with the parallelized GA optimizer. All runs started from the same initial guess previously generated. The solutions were checked to assure that each case converged to the same SNOPT optimality and feasibility criteria in the same number of iterations, n_i. All runs generated the same optimal separation time of 430.03 seconds. Furthermore, n_{dv} and n_l were constant. Comparisons were made on one machine, Lomax, which is part of the NASA Advanced Supercomputing (NAS)[21] facility located at NASA Ames Research Center. Table 3 shows some specifications for Lomax.

Figure 9 illustrates r_s vs. n_p for the series of parallel runs. Table 4 shows the parameters of the run that served as a baseline from which the r_s were computed. Figure 9 contains four curves. The curve labeled Equation 19 shows values of Eq. (19) for the baseline parameters and discrete n_p. Because n_l is slightly different in 0.5 and 0.25 time step cases, the two values were averaged. The baseline run was used as run 1 in the equation. There are two actual curves that were created using integration time steps of 0.5 and 0.25 seconds. They show the actual measured r_s relative to the baseline. The linear curve is the one that would result if the process was perfectly parallel. For example, a doubling of n_p would double r_s.

Figure 9 shows that increases of r_s decrease as n_p increases. After a certain n_p is reached, no significant increase in r_s is observed. Inspection of Eq. (19) reveals that the curves should be flat for all n_p greater than $n_{dv} + 1$, implying that greater speed increases can be achieved by increasing n_{dv}. The actual curves flatten out sooner than the Eq. (19)

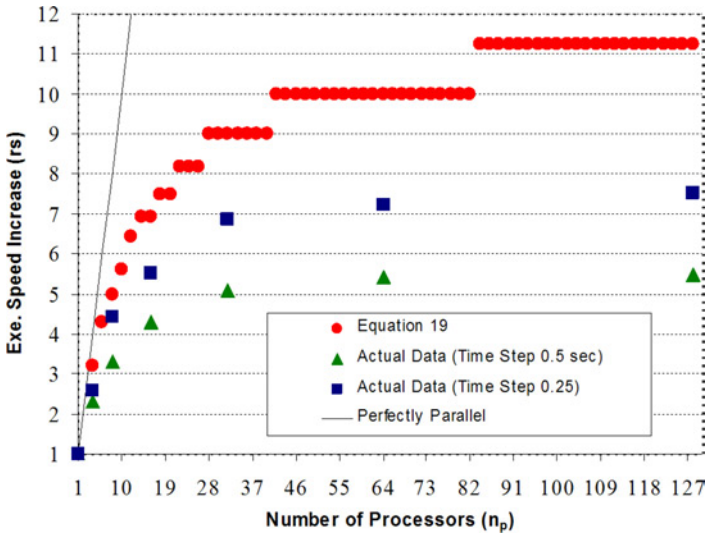

Fig. 9 Execution speed ratio vs. number of processors.

Table 4 Parameters of baseline run.

Time Step (sec)	n_p	n_{dv}	n_l	n_i	η_{opt} e-3	η_{feas} e-10	wall t (sec)
0.5	1	83	6.2	138	9.0	3.3	2580
0.25	1	83	5.7	100	4.5	720	3624

curve. This is because as n_p increases the program must spend more time setting up processors and exchanging data among processors. The relative time expense of these can be reduced by halving the integration time step. See that the 0.5 actual curve flattens out sooner than the 0.25 actual curve. However, the 0.25 case requires more execution time. Hence, increases in computational efficiency (i.e. effective use of all processors) are achieved at the expense of longer execution times. Similar effects would be observed if the time step was held constant and the trajectory duration increased.

VIII. Conclusion

Mission is designed to address the set-up and operating issues of RLV trajectory design and optimization within AEE. Specifically, *Mission* simplifies set-up of data exchange between tools by using XML "marked up" input data. Its object-oriented C++ code design allows for convenient extension of mathematical models and functions. *Mission* realizes autonomous operation through the use of a GA-based optimizer, and it achieves reduced execution times through parallelization. An approximate factor of 7 speed increase was observed for 32 processors and a time step of 0.25 seconds for the example case solved here. Greater speed increases and increases in computational efficiency can be achieved by making the problem larger (i.e. increasing n_{dv}, decreasing the integration time step, or increasing the duration of the trajectory).

Future research and development of selection strategies could improve performance of the GA-based optimizer. In addition, more effort could be spent on optimizing the sequencing of message passing between processors. Progress here would result in actual execution time ratios moving closer to those predicted by Eq. (19).

Acknowledgments

The authors acknowledge Dr. John Melton, NASA Aerospace Engineer, for writing the GA optimizer used in this study and Ivan Peragine, NASA student intern, for executing the many computer runs necessary for compiling data and generating the parallel processing figures shown in this paper.

References

[1]Monell, D., Mathias, D., Reuther, J., and Garn, J., "Multi-Disciplinary Analysis for Future Launch Systems Using NASA's Advanced Engineering Environment," AIAA 2003-3428, June 2003.

[2]Windhorst, R., Ardema, M., and Bowles, J., "Minimum Heating Entry Trajectories for Re-usable Launch Vehicles," *Journal of Spacecraft and Rockets*, Vol. 35, No. 5, 1998, pp. 672–682.

[3]Saunders, D., Allen, G., Gage, P., and Reuther, J., "Crew Transfer Vehicle Trajectory Optimization," AIAA 2001-2885, June 2001.

[4]Brauer, G., Cornick, D. E., and Stevenson, T., "Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)," NASA CR-2770, Feb. 1977.

[5]Hargraves, C. R., and Paris, S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *Journal of Guidance, Control, and Dynamics*, Vol. 10, No. 4, 1987, pp. 338–342.

[6]Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice-Hall, Inc., New Jersey, 1991, pp 9–10.

[7]Miele, A., *Flight Mechanics, Theory of Flight Paths*, Addison-Wesley Publishing Company, INC., Palo Alto, 1962, pp. 58–66.

[8]*U.S. Standard Atmosphere, 1976*, NOAA, NASA, and USAF, U.S. Government Printing Office, Washington D.C., 1976.

[9]Betts, J. T., "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193–207.

[10]Gill, P. E., Murray, W., and Saunders, M., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal of Optimization*, Vol. 12, No. 4, pp. 979–1006.

[11]Gage, P., "New Approaches to Optimization in Aerospace Conceptual Design," NASA Ames Research Center, Contractor Report 196695, Moffett Field, CA, March 1995.

[12]Beasley, D., Bull, D., and Martin, R., "An Overview of Genetic Algorithms: Part 1 Fundamentals," *University Computing*, Vol. 15, No. 2, 1993, pp. 58–69.

[13]Harold, E. R., and Means, W. S., *XML in a Nutshell*, O'Reilly and Associates, INC., Sebastopol, 2001, pp. 1–8.

[14]Apache Website, http://xml.apache.org/xerces-c/.

[15]Vander Kam, J., and Gage, P., "The Launch Vehicle Language (LVL) Data Model for Evaluating Re-usable Launch Vehicle Concepts," to be submitted to the AIAA Aerospace Sciences Meeting, Reno, 2004.

[16]Lau, E., Wong, J., Vander Kam, J., Jiang, X., Palmer, G., and Gage, P., "LVL Editor: A Graphical Tool for Visualizing and Managing Data in an Engineering Component Hierarchy," AIAA-2004-1173, Jan. 2004.

[17]Jiang, X., and Gage, P., "Weights Analysis of Space Launch Vehicles in an Advanced Engineering Environment," to be submitted to the AIAA Aerospace Sciences Meeting, Reno, 2004.

[18]Argonne National Laboratory Website, http://www-unix.mcs.anl.gov/mpi/

[19]Stone, H. W., and Piland, W., "21st Century Space Transportation System Design Approach: HL-20 Personnel Launch System," *Journal of Spacecraft and Rockets*, Vol. 30, No. 5, 1993, pp. 521–528.

[20]Kontinos, D. A., Gee K., and Prabhu, D., "Temperature Constraints at the Sharp Leading Edge of Crew Transfer Vehicle," AIAA Paper 2001–2886.

[21]NAS Website, http://www.nas.nasa.gov/.