AIAA 2000-4884
Optimized Solutions for the Kistler *K-1* Branching Trajectory Using MDO Techniques

L. A. Ledsinger
J. R. Olds
Space Systems Design Laboratory
Georgia Institute of Technology
Atlanta, GA

8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization

**6-8 September 2000
Long Beach, California**

Optimized Solutions for the Kistler *K-1* Branching Trajectory Using MDO Techniques

Laura A. Ledsinger[†]
Dr. John R. Olds[††]
Space Systems Design Laboratory
School of Aerospace Engineering
Georgia Institute of Technology, Atlanta, GA 30332-0150

ABSTRACT

Fully reusable two-stage-to-orbit vehicle designs that incorporate 'branching' trajectories during their ascent are of current interest in the advanced launch vehicle design community. Unlike expendable vehicle designs, the booster of a reusable system must fly to a designated landing site after staging. Therefore, both the booster return branch and the orbital upper stage branch along with the lower ascent trajectory are of interest after the staging point and must be simultaneously optimized in order to achieve an overall system objective. Current and notable designs in this class include the U. S. Air Force Space Operations Vehicle designs with their 'pop-up' trajectories, the Kelly Astroliner, the Kistler *K-1*, the two-stage-to-orbit vehicle *Stargazer*, and NASA's proposed liquid flyback booster designs (Space Shuttle booster replacement).

The solution to this problem using an industry-standard trajectory optimization code (POST) typically requires at least two separate computer jobs — one for the orbital branch, from the ground to orbit, and one for the flyback branch, from the staging point to the landing site. These jobs are coupled and their data requirements are interdependent. These requirements must be taken into consideration when optimizing the entire trajectory. This paper analyzes the results of branching trajectory optimization for the Kistler *K-1* launch vehicle with respect to computational efficiency and data consistency for various solution methods. In particular, these methods originate from the field of Multidisciplinary Design Optimization (MDO).

[†] - *Ph.D. Candidate, School of Aerospace Engineering, Member AIAA.*

[††] - *Assistant Professor, School of Aerospace Engineering, Senior Member AIAA.*

NOMENCLATURE

NASA	National Aeronautics and Space Admin.
RTLS	Return to Launch Site

INTRODUCTION

In order to lower costs, designers of advanced two-stage-to-orbit (TSTO) launch vehicles are considering launch systems in which the booster stage can be recovered, serviced, and reflown. Often the reusable booster is required to land at a predesignated recovery site either near the original launch site (RTLS-style trajectory, Figure 1) or downrange of the staging point (Figure 2). In these cases, the entire trajectory is composed of three parts. The ascent portion follows the vehicle from launch to staging. At this point, the trajectory is assumed to split into two 'branches.' One is the *orbital branch* beginning at staging and following the orbital upper stage all the way to orbit. The second branch, or *flyback branch*, starts at staging and follows the reusable booster to its landing site. Due to recovery distance or out-of-plane maneuvers required, the booster is often powered for its flight to the landing site. In simulations where the booster is jettisoned from an orbital vehicle, it may be convenient to combine the ascent trajectory and the orbital branch to create one computer job, as is the case with the Kistler *K-1*. The same may be said about a launch vehicle with an upper stage that is jettisoned; the ascent trajectory and the flyback branch may be combined.

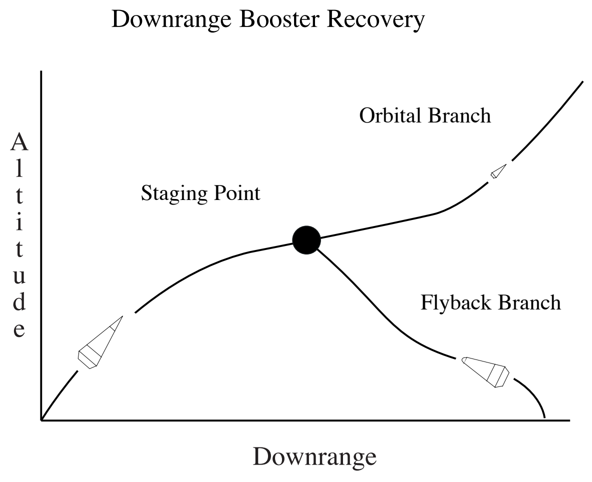

Figure 1: RTLS Branching Trajectory

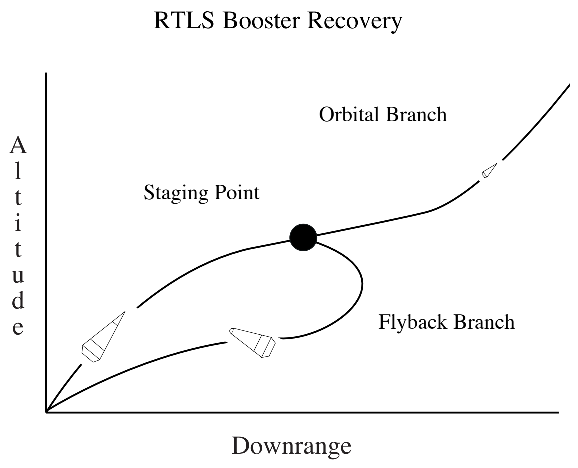

Figure 2: Downrange Branching Trajectory

In general, both the orbital branch and the flyback branch rely upon the ascent trajectory for their respective initial conditions. These initial conditions are state vectors composed of geographical position, altitude, velocity, flight path angle, velocity azimuth, and possibly staging weight. The ascent trajectory also depends on both branches. Assuming that the booster is powered, the amount of flyback fuel required by the booster influences the gross lift-off weight of the vehicle and thus the ascent path. The weight of the upper stage (which is dependent upon initial staging conditions) also affects the gross lift-off weight of the vehicle and thus the ascent path. Consequently, all the parts of the entire trajectory are coupled or interdependent.

The optimization of branching trajectories differs a great deal from that of the single trajectory. The fact that there are now two, or even three, different parts of the overall branching trajectory makes the optimization more complex. The existence of the staging point means that compromises must be made between the orbital and flyback branches. An example is that typically the upper stage wants a larger flight path angle at staging. This helps it achieve its orbit goals in a shorter amount of time (than with a smaller flight path angle) and thus aids in minimizing its fuel consumed during the orbital portion of the trajectory. At the same time, the booster desires a smaller flight path angle. The closer the velocity vector is to the horizontal, the faster the booster can achieve that negative flight path angle that is needed to aim the vehicle back to the earth. This also helps to minimize fuel. It is evident that a compromise is needed when the overall trajectory is considered.

Another important aspect of the branching trajectory is the feedback of the flyback fuel and the upper stage fuel. Unnecessary extra fuel in the booster or the upper stage means that either extra payload can be taken to orbit or smaller vehicles (booster and upper stage) can be used. Not enough fuel means that orbit conditions may not be met and the booster does not return to its designated landing site. Thus, the feedback of these fuels is required.

Traditional Solution Methods

Unfortunately, a common method currently used in industry for optimizing a branching trajectory problem (henceforth the *'One-and-Done' Method*), while recognizing the coupling of the ascent trajectory and orbital branch, ignores the flyback fuel dependency from the flyback branch to the ascent trajectory. The ascent trajectory, orbital branch, and flyback branch are treated as separate, but sequential optimization subproblems. A reasonable guess at upper stage mass, flyback fuel, and associated structure is made to establish an initial booster weight. Then, the ascent is optimized for maximum weight at staging (or some other similar criteria). The ascent trajectory will produce a staging state vector used to initiate the orbital branch and the flyback branch. This vector includes altitude, velocity, flight path angle, velocity azimuth, latitude, longitude, and sometimes staging weight. The orbital branch will typically be optimized with respect to maximizing the upper stage burnout weight, while the flyback branch will typically be optimized with respect to minimizing the flyback fuel consumed.

There are a number of deficiencies in the 'one-and-done' method. A major deficiency is that the final solution is not 'internally consistent,' in other words, it is not guaranteed to be converged between the subproblems. The feedback is not there; the problems that this creates have been discussed above.

This aforementioned deficiency can be eliminated through iteration between the ascent, upper stage, and the flyback branches. From this point on, this method will be referred to as the Manual Iteration method. A significant deficiency still exists with this method as with the 'one-and-done' method.

At a fundamental level, these methods are inherently flawed. The objective functions of the subproblems are not the same; therefore, they can be in conflict. If the system-level objective is to deliver a certain payload to orbit with a minimum weight booster, then why expect an optimum solution from a method that first maximizes the payload to orbit for the orbital branch, then minimizes the flyback fuel for the flyback branch? A compromise in the staging conditions can be made such that it reduces the flyback fuel and thus decreases the booster weight. A proper solution to this problem requires simultaneous and coupled treatment of all branches of the trajectory, and the establishment of a single, consistent objective function between them (i.e. a system-level optimization).

Many in industry have recognized the deficiencies of the 'one-and-done' and manual iteration methods. Some have employed optimizers that solve the branching trajectory problem as a whole. Boeing uses one OTIS[1] simulation for the entire branching trajectory. Shuttle-IUS branching trajectories have been simulated at the Aerospace Corporation[2] with a system-level optimizer. Feedback of the fuels was not considered.

At NASA Langley, research with POST (The Program to Optimize Simulated Trajectories) is continuing. POST I[3] can not simulate branching trajectories. Branching trajectory research has included investigations of a bimese vehicle[4] with a glideback return to the launch site and a Sänger-like vehicle[5] which included a powered RTLS. Such research with POST I did not include feedback or system-level optimizers. POST II[6] is currently being tested. POST II can simulate multiple vehicles and thus branching trajectories as one entire trajectory simulation. At this time, fuel feedback is not an option.

For this research, the optimization of branching trajectories has been solved using the POST I code. The solution methods of the 'one-and-done' and manual iteration methods rely on at most three separate POST input decks — one for the ascent to staging trajectory subproblem, one for the orbital branch subproblem, and one for the flyback branch subproblem. Each subproblem has its own independent variables, constraints, and objective function. The current research has retained the POST I code and the use of at most three separate input decks (one job for each part), but also eliminated any objective function conflict and lack of data consistency between them. This has produced a solution that resulted in internally consistent data (the fuels' feedback is reflected in the initial gross weight, etc.) and a single system-level objective function (without conflicting objective functions for each subproblem).

Trajectory Optimization with POST

POST I, typically used in conceptual design, is not capable of simultaneously treating and optimizing all parts of a branching trajectory. It is a Lockheed Martin and NASA code that is widely used for trajectory optimization problems in advanced vehicle design. POST is a generalized event-oriented code that numerically integrates the equations of motion of a flight vehicle given definitions of aerodynamic coefficients, propulsion system characteristics, atmosphere tables, and gravitational models. Guidance algorithms used in each phase are user-defined. Numerical optimization is used to satisfy trajectory constraints and minimize a user-defined objective function by changing independent steering and propulsion variables along the flight path. POST runs in a batch execution mode and depends on an input file (or input deck) to define the initial trajectory, event structure, vehicle parameters, independent variables, constraints, and objective function. Multiple objective functions and simultaneous trajectory branches cannot currently be defined in POST I.

For this research, the optimization of branching trajectories has been solved using the POST I code. The solution methods of the 'one-and-done' and manual iteration methods rely on at most three separate POST input decks — one for the ascent trajectory subproblem, one for the orbital branch subproblem, and one for the flyback branch subproblem. Each subproblem has it's own independent variables, constraints, and objective function. This research has retained the POST I code and the use of at most three separate input decks (one job for each part), but also eliminated any objective function conflict and lack of data consistency between them. This has produced a solution that resulted in internally consistent data (the fuels' feedback is reflected in the initial gross weight, etc.) and a single system-level objective function (without conflicting objective functions for each subproblem).

THE KISTLER *K-1*

To provide applicability to this research, the missions of candidate TSTO launch vehicle designs were chosen to serve as reference missions. In this paper, the analysis and results for the Kistler *K-1* is presented.

Kistler K-1

Many launch vehicles are currently being developed by commercial industries with the goal of capturing a profitable share of the growing satellite launch market. Such is the case of the Kistler *K-1* launch vehicle.[7] The *K-1* (Figure 3, Ref. 8) will be a fully reusable, two stage vehicle that incorporates branching trajectories. The vehicle's booster will use three Aerojet modified NK33 engines and the upper stage will be propelled by one Aerojet- modified NK43 engine.[9,10] There will be different versions of the vehicle to accommodate various payload classes. One of its missions will be to deliver a 3400 lbs to a 51 nmi x 486 nmi x 51° orbit. This is the mission that will be analyzed in this study. The data (weights, trajectory constraints, engine data, etc.) pertaining to that mission was provided directly by Kistler Aerospace.[11]

Figure 3: The K-1 *Launch Vehicle*[8]

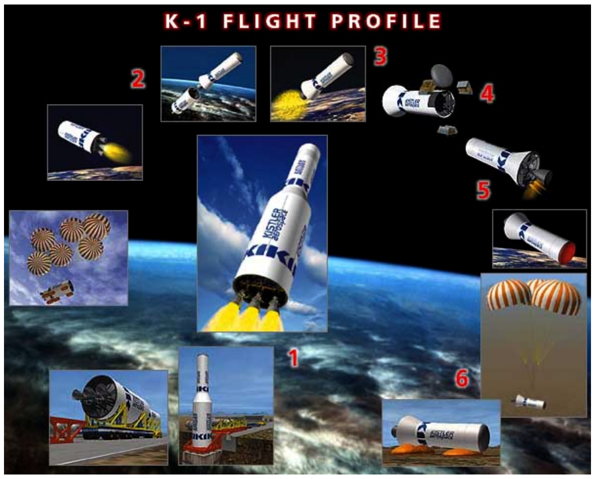

Figure 4: The K-1 *Trajectory*[8]

The trajectory of the *K-1* launch vehicle can be seen in Figure 4.[8] It will consist of an RTLS type branching trajectory as in Figure 1. After launch from the site at Woomera, Australia, the entire *K-1* will fly until staging approximately 120 seconds later. After staging, the booster performs a pitcharound maneuver that will guide itself back to within 10,000 ft of the launch site, to land with airbags and parachutes. The upper stage will continue on to the designated orbit. For the purposes of this study, the simulation will end when the orbital targets have been attained. In reality, the *K-1*'s upper stage will deorbit and return to the launch site.

The trajectory is simulated through two POST decks. The first follows the vehicle from launch to orbital injection of the upper stage. Note that this specific simulation combines the ascent and orbital paths. The second, or flyback branch, follows just the booster from staging to its return to the launch site.

The reference *K-1* ascent trajectory deck's independent variables are twelve pitch angles and payload weight. The ascent has five constraints including orbital insertion criteria and will maximize the payload for a given set of propulsion characteristics, vehicle aerodynamics, *K-1* weights, and ascent propellant.

The reference flyback trajectory deck uses six independent variables: four pitch angles, azimuth of the pitcharound maneuver needed to initially head the vehicle in a direction back to the launch site, and engine burn time. The two constraints guarantee a smooth rocket pull-up and landing within a certain downrange distance. Given a set of engine propulsion characteristics, aerodynamics, and a staging point, the flyback trajectory nominally tries to minimize flyback fuel weight. The required staging point data from the ascent branch includes altitude, flight path angle, latitude, longitude, velocity, and velocity azimuth.

SOLUTION APPROACH

The goal of the research was to retain the current analysis tool (POST) while producing a solution that results in internally consistent data (the true booster flyback fuel is reflected in the initial gross weight, etc.) and a single system-level objective function (without conflicting objective functions for each subproblem). In addition, the solution should be reasonably fast, robust, and efficient.

Solution techniques from the field of Multidisciplinary Design Optimization (MDO) were advantageously applied to the branching trajectory problem as it is posed as a coupled set of subproblems. Table 1 lists the characteristics of the MDO solution techniques that were used— fixed-point iteration (FPI), two variations of optimization-based decomposition (OBD), and collaborative optimization (CO). (POBD stands for partial optimization-based decomposition, in which only the feedback loops are broken. FOBD stands for full optimization-based decomposition, in which both feedforward and feedback loops are broken, resulting in a completely parallel execution.) In addition, an entry labeled 'Manual Iteration' is included for comparison. These methods have been used successfully by others for preliminary aircraft design[12] and launch vehicle design.[13] The FPI method is a serial execution technique that uses an overall system optimizer. This method explicitly uses the coupled feedforward/feedback loops linking the variables of the subproblems. The collaborative and parallel optimization methods are decomposition algorithms in that they break feedback/feedforward loops between the subproblems and incorporate an overall system optimizer. In addition, collaborative optimization is a multi-level optimization scheme. These techniques are detailed in Reference 14 for the general branching trajectory problem.

Note that there are many ways to optimize the trajectories of both the upper stage and the booster. In this research, all of the methods analyzed for the *K-1* had a system-level objective of maximizing payload weight. For the *K-1* simulation, fixed weights were used for all weights except for booster ascent propellant weight, flyback fuel weight, and payload weight. The constant total propellant weight was the sum of the booster ascent propellant weight and the flyback fuel weight used. The payload weight was the objective to be maximized.

Table 1: Solution Techniques to Branching Problems

Method	Internally Consistent Data	Iteration Between Branches	Conflicting Objective Functions	System-level Optimizer	Analysis Execution	Optimizer Strategy
Manual Iteration	Yes	Yes	Yes	No	Sequential	Distributed
Fixed-Point Iteration (FPI)	Yes	Yes	No	Yes	Sequential	System Level (large)
Partial OBD	Yes	No	No	Yes	Sequential	System Level (very large)
Full OBD	Yes	No	No	Yes	Parallel	System Level (extremely large)
Collaborative	Yes	No	No	Yes	Parallel	Distributed

RESULTS FOR THE KISTLER *K-1*

'One-and-Done' Method

The 'One-and-Done' method does not account for the iterative, coupled nature of the ascent and flyback branches. The data extraction/insertion from the ascent POST deck to the flyback deck was performed manually. The results for this method appear in Table 2. The solution for this method will be the starting point for all the methods following this one. As a result, computational time is not listed for this method. The main reason to show this method's results is to see the large difference in the objective function (recall that the goal is to maximize payload weight) that can be achieved when iteration occurs.

Table 2: 'One-and-Done' and Manual Iteration Method Results for K-1

	Payload Weight (lbs)	POST CPU Time	Iterations
'One-and-Done'	3,315	-	0
Manual Iteration	3,529	201 sec	6

For this method, the initial guess for booster ascent propellant is significant. The percentage of booster ascent propellant with respect to the total available used for this simulation was 92.88%. This left a little more than 7% for the flyback fuel. After the serial execution of the two POST decks, it was found that 1.2% of the initial flyback fuel was left over, or not used. If the guess for booster ascent propellant percentage was too high, then the possibility of not having enough flyback fuel would have existed. In that case, as far as the flyback simulation is concerned, the constraints would have been met, however, negative propellant would be used. In other words, the POST deck would have used the dry weight as propellant, resulting in an obviously wrong answer. That scenario highlights an example of one of the many deficiencies of this method.

As stated previously, the next method and the MDO methods will all begin with the solution to the 'one-and-done' method. Thus, the initial guesses are as follows: percentage of booster ascent propellant – 93.92%, percentage of flyback fuel – 6.08%, and payload weight – 3,314.79 lbs.

Manual Iteration Method

The manual iteration method uses two subproblem optimizers and no system-level optimizer. Execution is sequential and iterative between the ascent deck and the flyback deck. The flyback weight is updated as the iterations occur. Again, the data extraction/insertion is manual. Iteration information and execution time results are shown in Table 2. For this case, iteration was performed between the two basic subproblems to ensure data consistency (unlike the 'one-and –done' method), however the conflicting objective functions were not addressed. The convergence criterion for the manual iteration method was flyback fuel weight. The *K-1* was considered converged when this variable came within 0.01% of the result from the previous iteration. This result will be used as a comparison case in the MDO method assessment currently being conducted.

MDO Methods

The MDO methods of fixed-point iteration, optimization-based decomposition, and collaborative optimization all require a system-level optimizer. The size of that optimizer for each of the MDO methods is listed in Table 3. The system-level optimizer used for the MDO methods was the Modified Method of Feasible Directions implemented by the software program DOT™. For the FPI and OBD methods, the gradients for the system-level optimizer were calculated using central finite differences. The entire process for each MDO method, including data extraction/insertion and gradient calculation, was automated using PERL and C++ codes.

Table 3: Size of System Optimizer for the K-1 Branching Case

	Kistler *K-1*	
	Design Variables	Constraints
FPI	19	10
POBD	20	11/12
FOBD	26	24
CO	7	2

DOT™ requires that equality constraints be formulated as inequality constraints. Three of the constraints for the *K-1* POST decks were equalities; formulated as two inequality constraints, this brings the total number of constraints for the FPI method to ten. Note that for the partial OBD method there were either one or two more constrains in addition to those of the FPI method. The compatibility constraint can be posed as either a squared inequality constraint (eleven total constraints) or two inequality constraints (twelve total constraints.) The compatibility constraints for the full OBD method were posed as seven pairs of inequality constraints, thus the total number of constraints was twenty-four.

Fixed-Point Iteration

The FPI method involves use of a system-level optimizer and POST deck iteration. The POST decks are not optimized for this method. They are used to integrate the equations of motion using the set of controls given by the system-level optimizer. The POST deck iterations are considered converged when the flyback fuel weight was within 0.01% of itself, just like in the manual iteration method.

Detailed quantitative results can be seen in Tables 4 & 5. The FPI method gave an optimized solution of 3,544 pounds of payload weight in 16.3 minutes with eighteen system-level iterations. Figure 5 shows how the payload weight varied with function call. In addition, the plot shows the number of MMFD iterations, the line searches needed to define the search direction, per function call. Figure 6 shows the active constraint history for the FPI method, active constraints being those that were either violated or greater than –0.5.

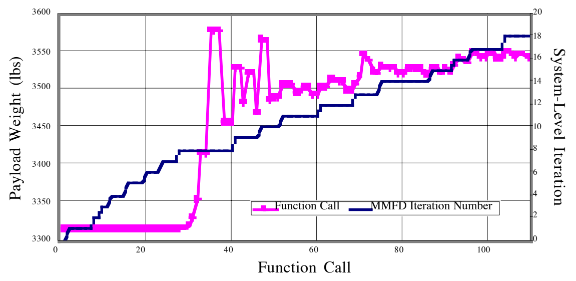

Figure 5: Payload Weight and System-Level Iterations per Function Call for the FPI Method

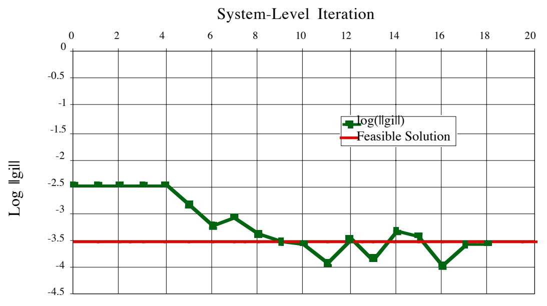

Figure 6: Active Constraint History per System-Level Iteration for the FPI Method

Optimization-Based Decomposition

Like the FPI method, the Partial OBD method involves use of a system-level optimizer and POST decks are not optimized. The feedback loops are eliminated by this method. The POBD methods resulted in an optimized payload weight of 3,567 pounds in twenty system-level iterations requiring 15.7 minutes. Additional numerical results are listed in Tables 4 & 5.

The Full OBD method also involves the use of a system-level optimizer and POST decks that are not optimized. For this method, there is no feedback *or* feedforward and the POST decks are executed in parallel. For the FOBD method, the optimization had to be restarted once due to a lack of progress the first time. After this restart, an optimal solution of 3,585 pounds of payload weight was found. This took about sixteen minutes total in thirty-four system-level iterations for the entire problem. More quantitative results are listed in Tables 4 & 5.

Collaborative Optimization

Table 4: K-1 Results Comparison (MDO)

	Ascent Deck	Flyback Deck
Flyback Fuel Weight	Input	Output
Altitude	Output	Input
Velocity	Output	Input
Azimuth Velocity	Output	Input
Flight Path Angle	Output	Input
Latitude	Output	Input
Longitude	Output	Input

The CO method involves the use of a system-level optimizer and a parallel analysis structure. However, for this multi-level decomposition scheme, the POST decks *are* optimized using the NPSOL optimizer included in the POST software. There were eight target variables required for this method. These included the payload and flyback fuel weights and the six variables that composed the staging vector. Table 4 shows how these target variables were perceived by the ascent and flyback POST decks, as either inputs or outputs. The system-level constraints, or J's, were calculated accordingly.

Constraint gradient calculations were performed using the post-optimality sensitivity analysis.[15] The benefit from this analysis, in that numerous analysis calls can be eliminated, is exploited if the objective function gradient can also be calculated analytically. This was achieved by adding the objective function, payload weight, to the vector of target variables. In addition, it was included in the error, J, for the ascent deck and was perceived as an output *from* the ascent deck (but an input/control *in* the ascent deck). Consequently, the objective function gradient was easily and analytically derived.

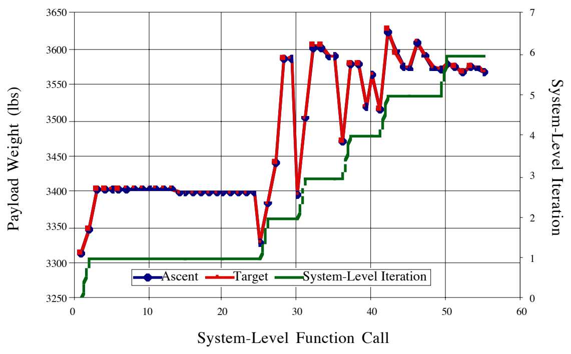

Figure 7: Payload Weight Tracking

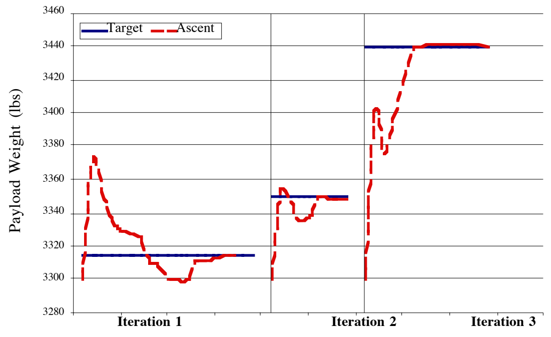

Figure 8: System-Level Coordination for Payload Weight

The CO method gave an optimal solution of 3,569 pounds of payload weight. This was attained in seven system-level iterations requiring 1.84 hours. More results are given in the next section. Figure 7 illustrates how the payload weight changed per system-level function call. Figure 8 shows the subsystem level target achievement for the first three system-level iterations (0, 1, and 2) for the payload weight. In Figure 8, it is shown that the payload weight increased at every iteration. This occurred because the ascent deck optimization matched the target at each iteration.

SUMMARY OF RESULTS

Tables 5 and 6 give quantitative results for the Kistler *K-1* MDO methods. In Table 4, POST computational time refers to the total amount of time to run the POST decks for all calls, including function calls and gradient calls.

As indicated in Table 5, all methods increased the payload weight of the *K-1*, the fixed-point iteration method by about 15 pounds. The partial optimization-based decomposition improved the payload weight of the *K-1* by approximately 23 pounds over the FPI method. The full optimization-based decomposition results in about a 42 pound increase in the payload weight over the FPI method. Use of the collaborative optimization method increased the payload by approximately 32 pounds. The decomposition methods gave an approximate 40 – 55 pound increase over the manual iteration method, a 1.6% increase, which is relevant considering the high costs per pound to launch payloads.

Table 5: K-1 Results Comparison (MDO)

Method	Optimized Payload Weight	POST Computational Time
FPI	3,543.64 lbs	976.8 sec (16.3 min)
POBD 1	3,566.82 lbs	939.5 sec (15.7 min)
POBD 2	3,566.85 lbs	941.4 sec (15.7 min)
FOBD	3,585.06 lbs	1,808.5 sec (30.14 min)
CO	3,569.04 lbs	6,627 sec (1.84 hrs)

It was expected that the FPI, POBD, FOBD, and CO methods result in the same objective function; however, the lack of performance by the FPI method may be attributed to the numerical noise introduced by the flyback fuel weight convergence tolerance on the internal POST iterations. This noise also affects the gradient calculations for the engine-on time design variable because for that gradient, internal POST iterations occur. This tolerance then affects the flyback fuel weight and consequently the optimized payload weight. The marked difference in the FOBD method may partially be attributed to the existence of tolerances for the staging vector since the feedforward no longer existed. These tolerances would have affected the final payload weight.

Some definitions for the columns in Table 6 are required. The third column, 'POST calls,' refers to how many times an analysis evaluation occurred, including system-level, gradient (except for the CO method), and line search evaluations. For the FPI and POBD methods one 'POST call' is a sequential execution of both POST decks. For the FOBD and CO methods, one 'POST call' is a parallel execution of the two POST decks. Note that there are two numbers given for the CO method. The first is the number of 'POST calls' for the ascent deck, the second is for the flyback deck. As would be expected the number of 'POST calls' for the collaborative optimization method is significantly larger than that for the other methods. While the number of system-level iterations is smaller for CO, many function calls with the subsystems at each iteration were required.

Table 6: K-1 Detailed Results Comparison (MDO)

Method	POST Calls	System-Level Iterations	Avg. CPU Time/Function Call
FPI	884	18	1.105 sec
POBD 1	843	20	1.114 sec
POBD 2	843	20	1.117 sec
FOBD	1622	34	0.911 sec
CO	2271/4745	7	1.326 min

In the fourth column, CPU time refers to the average time it took for one POST call to run, be that iteratively, sequentially, or in a parallel manner. At first glance, the average CPU times per function call listed in Table 6 are not what one would expect. The average time for the FPI method is usually larger than that for the OBD methods because the internal iterations for convergence are many. For the Kistler case, internal convergence occurred in zero or one iteration, usually zero for gradient calculation (this is because the flyback fuel weight is controlled by 'engine-on' time, one of the design variables). Thus for the majority of 'POST call', one POST call for the

FPI method would take approximately the same amount of time as one POST call for the POBD method. This is the case as shown in Table 6. The time for the FOBD method was smaller since the POST decks are executed in a parallel manner. The Ascent POST deck required a longer amount of execution time than the Flyback deck and its average time is that listed in Table 6. Even though the POST decks for the CO method were executed in parallel, the CPU time was longer because optimization of the POST decks required a longer amount of time than a simple integration of the equations of motion did.

Note that the results for the POBD method were the same regardless of which way the compatibility constraints were posed. This was not the case, however, for the FOBD method. When the compatibility constraints were formed as one inequality per design variable, a feasible solution was not found.

Table 7: Staging Data Results for the K-1

Method	Altitude (ft)	Velocity (ft/sec)	Gamma (deg)
MIM	138,028	4,172.28	33.337
FPI	138,019	4,169.27	33.520
POBD 2	138,014	4,166.92	33.272
FOBD	137,542	4,160.34	32.892
CO	138,028	4,171.15	32.667

More quantitative results can be seen in Table 7 which shows the difference in staging vectors for the methods. Included for comparison is the staging vector for the manual iteration method. Since the results for the two POBD methods were almost exactly the same, just one is included in the table. Assuming that convergence tolerances affect the FPI method staging vector, the results from the other MDO methods imply that a smaller flight path angle can reduce flyback fuel weight consumed and thus, the increase the payload weight. (In the FOBD case, the lower staging altitude also has a similar effect.) The returning booster desires this lower angle so that it can perform its pitcharound maneuver more efficiently.

Through the ascent, the trajectories are very similar with regards to altitude and velocity. In fact, as can be seen in Table 7, the staging points are relatively close to one another. The angles of velocity azimuth, latitude, and longitude are expected to be identical considering there are no yawing movements during the ascent and the results of the table recognize that fact. In Figure 8, it is shown that the flyback altitudes exhibit differences. Differing pitch angles during the flyback pitcharound maneuver affect the flyback trajectory seen in Figure 9 as does the initial altitude.

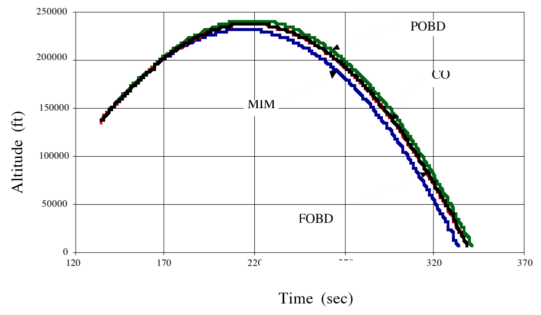

Figure 9: Altitude versus Time for the K-1 *Branching Trajectory*

CONCLUSIONS

This paper has provided an introduction and results to the Kistler *K-1* branching trajectory optimization problem. The main deficiencies of the traditional methods of the 'one-and-done' and manual iteration methods were discussed for the branching trajectory problem. The main deficiency was that conflicting objective functions existed in these methods. Each POST deck had its own objective to fulfil, but compromises in each individual trajectory could benefit the entire trajectory. Multidisciplinary design optimization methods introduced a system-level optimizer that resulted in an overall, system-level objective being met for the branching trajectory problem. The use of these methods for the Kistler *K-1* problem showed that an increase in payload weight of 1%, on average, could be obtained. Is this rather small percentage worth it? When one considers the costs of launching payloads, about $5,000 - $6000 per pound, yes. More payload can be put into orbit per launch and thus, more profits.

The MDO techniques used to solve the branching trajectory problem were Fixed-Point Iteration, with a single system-level optimizer; Optimization-Based Decomposition to eliminate iteration between the branches (two different formulations); and Collaborative Optimization to enable parallel subproblem execution with distributed, coordinated optimizers. The results of solving the *K-1* branching trajectory with MDO methods indicated a trend from the manual iteration method. These results showed that a decrease in the flight path angle at staging aided in reducing the booster's flyback propellant. This allowed more payload weight to flown to orbit.

ACKNOWLEDGEMENTS

The authors would like to thank the members of the Space Systems Design Lab (SSDL) at the Georgia Institute of Technology for their support. Thanks to Mr. Dick Kohrs of Kistler Aerospace for his assistance with the *K-1* data.

This work was partially sponsored by NASA via the Georgia Space Grant Consortium and Marshall Space Flight Center.

REFERENCES

1. Paris, Steve, "Overview of OTIS 3.0," NASA Conference Publication No. 10187, August, 1996.

2. Hallman, W. P, "Mission Timeline for a Shuttle-IUS Flight Using a Nonstandard Shuttle Park Orbit," TOR-0083 (3493-14)-1, The Aerospace Corporation, October, 1982.

3. Brauer, G. L., D. E. Cornick, and R. Stevenson, "Capabilities and Applications of the Program to Optimize Simulated Trajectories," NASA CR-2770, Feb. 1977.

4. Naftel, J. C. and R. W. Powell, "Flight Analysis for a Two-Stage Launch Vehicle with a Glideback Booster," *Journal of Guidance, Control, and Dynamics,* Vol. 8, No. 3, pp. 340-343, May – June, 1985.

5. Lepsch, R. A. and J. C. Naftel, "Winged Booster Performance with Combined Rocket and Airbreathing Propulsion," *Journal of Spacecraft and Rockets*, Vol. 30, No. 6, pp. 641-646, November – December, 1993.

6. Anderson, R. L., et. al. *Program to Optimize Simulated Trajectories (POST II): Guide for New Users*, Preliminary, Beta Release, Volume I, April, 1999.

7. Kohrs, R. and R. Petersen, "Development of the *K-1* Two-Stage, Fully-Reusable Launch Vehicle," AIAA-98-1540, April, 1998.

8. Kistler Aerospace web site: http://www.kistleraerospace.com.

9. Anisimov, V. S., T. C. Lacefield, and J. Andrews, "Evolution of the NK-33 and NK-43 LOX/Kerosene Engines," AIAA Paper 97-2680, July, 1997.

10. Mecham, Michael, "Aerojet Acquires Major *K-1* Engine Shipment," *Aviation Week and Space Technology*, Vol. 147, n. 10, September, 1997, pp. 61-62.

11. Kohrs, R., Personal Communications, June and December, 1999.

12. Kroo, I., S. Altus, R. Braun, P. Gage, and I. Sobieski, "Multidisciplinary Optimization Methods for Aircraft Preliminary Design," AIAA Paper 94-4325, 1994.

13. Braun, R. D., R. W. Powell, R. A. Lepsch, D. O. Stanley, and I. M. Kroo, "Multidisciplinary Optimization Strategies for Launch Vehicle Design," AIAA Paper 94-4341, September, 1994.

14. Ledsinger, Laura A. *Solutions to Decomposed Branching Trajectories with Powered Flyback Using Multidisciplinary Design Optimization.* Ph.D. Thesis. June, 2000

15. Braun, R. D., I. M. Kroo, and P. J. Gage. "Post-Optimality Analysis in Aerospace Vehicle Design," AIAA Paper 93-3932, August, 1993.