AIAA Atmospheric Flight Mechanics Conference and Exhibit
20 - 23 August 2007, Hilton Head, South Carolina

AIAA 2007-6393

Mars Science Laboratory Entry Optimization Using Particle Swarm Methodology

Michael J. Grant* and Gavin F. Mendeck†
NASA Johnson Space Center, Houston, TX, 77058

An attempt to improve upon the manual, time consuming traditional point design method for current reference trajectory design is discussed. A single objective particle swarm optimization (SOPSO) algorithm and a multiobjective particle swarm optimization (MOPSO) algorithm were developed. The SOPSO algorithm was used to validate the capability of applied swarming theory to Mars entry optimization. The MOPSO algorithm is an extension of SOPSO, providing the capability to generate Pareto fronts in the environment of competing objectives. The Pareto fronts generated by MOPSO provided quick insight into entry trajectory design characteristics that took years to understand through the traditional point design process. The optimal trade associated with the conflicting objectives of supersonic parachute deployment altitude, range error ellipse length, and g-loading were quantified in a visual environment. It is shown that the analysis of the Pareto front allows the user to make intelligent reference trajectory design decisions that could potentially span various disciplines in vehicle design. This work serves as a beginning of a fundamental departure from the traditional point design process that is manual, time consuming, and only allows the designer to evaluate a select few designs. This fundamental change in design allows the designer to analyze the set of best solutions via an automated, efficient, and global exploration of the design space using particle swarm methodology.

Nomenclature

COV	=	coefficient of variance
c_1	=	cognitive parameter
c_2	=	social parameter
μ	=	mean of objective function
$\bar{p}_i^k$	=	best position particle i experienced up to iteration k
$\bar{p}_{in}^k$	=	position of best neighbor of particle i at iteration k
r_1^k	=	cognitive random factor (0,1) at iteration k
r_2^k	=	social random factor (0,1) at iteration k
σ	=	standard deviation of objective function
$\bar{v}_i^k$	=	velocity of particle i at iteration k
w^{k+1}	=	velocity inertia at iteration $k+1$
$\bar{x}_i^k$	=	current position of particle i at iteration k

I. Introduction

HISTORICALLY, the design of trajectory and vehicle characteristics necessary to accomplish a specific mission has been performed using a manual, user intensive point design approach. When the evaluation of a solution requires the execution of a computationally intensive simulation, this traditional process is rather slow since it

* Graduate Research Assistant at the Georgia Institute of Technology and Graduate Co-op Student, Flight Design and Dynamics Division, NASA Johnson Space Center, AIAA Student Member.
† Aerospace Engineer, Ascent/Descent Dynamics Branch, Flight Design and Dynamics Division, NASA Johnson Space Center, Houston, Texas, AIAA Member.

requires the designer to manually change design variables to obtain a pseudo-optimal design. As the number of design variables increases, the ability of the designer to fully explore the design space in a manual fashion is prohibited. In order to speed up the process, an automated optimization routine could be used in place of the manual optimization performed by the designer. Typical methods of optimization that include gradient-based methods are not sufficient for typical problems with a large number of design variables for two major reasons. First, the gradient approach assumes that derivatives of the objectives may be approximated about any feasible point in the design space. When the execution of a simulation is required to obtain the objective values for a given design, no guarantee can be made regarding the continuity of the corresponding derivatives. Second, as the number of dimensions in the trade space increases, the computational time necessary to approximate derivatives about any feasible solution becomes prohibitively large.

The design of the Mars Science Laboratory (MSL) 2009 mission is no exception to the design challenges illustrated above. A substantial amount of manual work has been performed to gain an understanding of how the entry performance is influenced by various vehicle characteristics, guidance inputs, and trajectories using the traditional point-design approach. The knowledge gained over past years has been utilized to develop many pseudo-optimal point designs. MSL is a project that spans multiple NASA centers including the Jet Propulsion Laboratory (JPL), Langley Research Center (LaRC), and Johnson Space Center (JSC). The main focus of the work in the Descent Analysis Branch at JSC has been to optimize the entry flight path angle and reference trajectory used by a modified Apollo guidance during Mars entry to enable a safe, precision landing.[1] The use of non-traditional methods to automate the trajectory design process at NASA JSC, such as the single objective particle swarm optimization (SOPSO) and the multiobjective particle swarm optimization (MOPSO), were explored since both only require function evaluations, eliminating the need for gradient computations. Also, both algorithms have the potential to solve single and multiple objective problems using a minimum number of function evaluations. Both of these properties of the SOPSO and MOPSO algorithms result in an efficient exploration of the design space when compared to the traditional point design approach. The MOPSO algorithm provides the ability to perform an automated, global optimization and visualization of competing objectives, resulting in the beginning of a fundamental departure from the traditional point design approach. A generic discussion of both methods and the specific application to the MSL design process follows.

II. Single Objective Particle Swarm Optimization

A. Overview

Particle swarm optimization (PSO) was invented in 1995 by Kennedy and Eberhart while attempting to simulate the unpredictable movement of a bird flock.[2] The apparent ability of birds to rapidly find food (while having no previous knowledge of the location of food) is analogous to the desire of a designer to rapidly determine the location in the design space that results in an optimal design. The SOPSO algorithm searches the n-dimensional design space for the optimal solution in a similar manner as real flocks search the physical 3-dimensional space for food. Both the SOPSO and MOPSO algorithms utilize a population of individuals that forms the swarm. This population represents the set of test solutions in the design space that are evaluated during a given iteration. The swarm is randomly initialized throughout the design space in order to promote diversity into the initial guess. The swarm investigates the design space by performing a coordinated, intelligent search that is influenced by the cognitive awareness of each individual's most optimal location visited and the communication of each individual in the swarm. This sharing of information offers an advantage over other optimization algorithms. The location of the individuals of the population is modified in subsequent iterations as each individual in the swarm updates its design space velocity.

B. Particle Movement

The swarm utilizes memory and communication to perform an effective search of the design space for the optimal solution. The velocity of particle *i* at iteration *k+1* ($\bar{v}_i^{k+1}$) is updated for each dimension of the design space according to Eq. (1).[3]

$$\bar{v}_i^{k+1} = w^{k+1}\bar{v}_i^{k} + c_1 r_1^{k}\left(\bar{p}_i^{k} - \bar{x}_i^{k}\right) + c_2 r_2^{k}\left(\bar{p}_{in}^{k} - \bar{x}_i^{k}\right) \tag{1}$$

Inertial Component — Cognitive Component — Social Component

As can be seen, the velocity calculation is composed of three separate components. Each component serves a unique role in the movement of the swarm. Note that the apparent complicated and random motion of a swarm may be simply expressed in one simple equation. In order to promote diversity in the search of the design space, r_1^k and r_2^k are random values obtained from a uniform distribution between 0 and 1. These random values change each iteration and allow the cognitive and social components of the equation to vary in importance throughout the entire optimization. This emulates the observed randomness associated with real swarms.

1. Inertial Component

The first component establishes global vs. local searching of the swarm. In order to permit a global search, the inertia at iteration $k+1$ (w^{k+1}) is initially set to a high value (usually near 1.0) such that the first term usually dominates. This will ensure that each member of the swarm performs a global exploration of the design space and will not be greatly influenced by its cognitive best location or by communication with other individuals of the swarm. Without a period of high inertia, the swarm will collapse to a local optimum without adequately exploring the design space. After a sufficient global exploration of the design space, the inertia may be reduced in order to permit a transition to localized swarming of the (hopefully) global optimal solution. The set of iterations in which the algorithm should transition from a global search to a local search via a reduction in inertia is usually not clear since the designer usually lacks knowledge of the objective landscape.

At first, the inertia was explicitly related to the iteration number such that the inertia is reduced as the iteration count increases. Both linear and quadratic functions of inertia were explored. This method sometimes resulted in a premature transition from a global search to a local search before the design space has been sufficiently explored. This resulted in the swarm incorrectly identifying a local optimum as the global optimum. Ideally, the inertia would be governed such that it is reduced at the lowest iteration count possible while providing sufficient exploration of the global design space. A proper balance between convergence speed (to the global optimal solution) and quality of solution (actually locating the global optimal solution) may be performed using the coefficient of variance (COV). A subset of individuals that have the most optimal objective values in memory is monitored by the SOPSO algorithm. This subset of objective values stored in each individual's memory is used to compute the COV. The COV is defined in Eq. (2) where σ represents the standard deviation of the subset optimal values and μ represents the average optimal value of the subset.[4]

$$COV = \sigma / \mu \tag{2}$$

For difficult (bumpy) problems, the standard deviation will remain high for a larger number of iterations. This will be due to the fact that the most optimal objective values in memory are constantly changing during the initial exploration of a dynamic design space. After the values in the subset converge, the standard deviation will consequently decrease. Once the COV is reduced below a tolerance specified by the user, the inertia is geometrically reduced by multiplying the current inertia by a factor specified by the user. When this occurs, the swarm slowly transitions from a global search to a localized search. The process is repeated throughout the optimization process and the inertia geometrically approaches zero. When the inertia is close to zero, the swarm is then performing a localized search and is only influenced by the cognitive and social terms of Eq. (1). The specification of the COV tolerance permits the designer to set the aggressiveness of the swarm. A high COV results in a more rapid reduction in inertia, aggressively transitioning the swarm from a global searching mode to a local searching mode.

2. Cognitive Component

As each individual in the swarm searches the design space, it remembers the design space location corresponding to the most optimal objective value it alone has seen. This design space location is utilized by the cognitive component of Eq. (1), where $\bar{p}_i^k$ represents the most optimal design space location in memory (at iteration k) of the i^{th} individual in the swarm and $\bar{x}_i^k$ represents the current design space location of the i^{th} individual. The term $\left(\bar{p}_i^k - \bar{x}_i^k\right)$ denotes the vector displacement of the individual's current location in the design space to the location it remembers as the best solution. c_1 is a constant value set to 2. While the choice of this value is rather arbitrary, it has been widely accepted in swarm

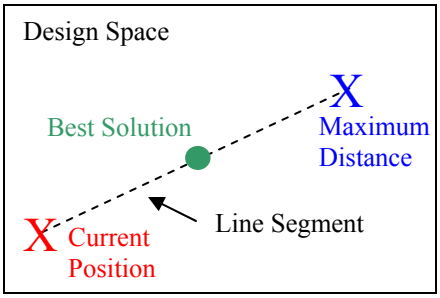


Figure 1: Cognitive Component Permissible Movement

optimization literature.[2,4] Also, it has a physical implication of how individuals search the design space. This c_1 value, in combination with the random value and vector displacement, permits the individual to travel anywhere on a specific line segment. This line segment begins at the current location of the individual and extends to a point on the opposite side and equidistant to the best location in memory (see Figure 1). Hence, this component provides a tendency for the individual to return to the region that it remembers is optimal.

3. Social Component

As mentioned before, a major advantage of SOPSO is the ability of the individuals in the swarm to communicate with one another during the search of the design space. Each individual of the swarm communicates with a number of neighbors specified by the user. The neighbors are determined by sorting the distance (in the design space) the other members of the swarm are from a given individual. Individuals that are closest serve as the set of neighbors for a given individual. Of the group of neighbors, the neighbor that has located the best solution serves as the social component of Eq. (1). Hence, the social influence is merely the influence placed upon an individual to move to the region of the design space of the best solution located by one of its neighbors, denoted as $\overline{p}_{in}^{k}$. Since c_2 is also a constant value of 2, the permissible movement associated with the social component is similar to that of the cognitive component.

4. Boundary Control

At high inertia values, individuals of the swarm will have a velocity that may potentially result in a violation of the design space boundaries set by the user. In such cases, the individual does not travel across the boundary but is instead placed at the boundary. The individual is then randomly chosen to either bounce or rest at boundary (the probability for each is 50%). If the individual is chosen to bounce, then the individual is placed on the boundary and its velocity is negated. If the individual is chosen to rest, then the individual is placed on the boundary and its velocity is set to zero in all dimensions. In certain cases, the swarm appeared to be more effective with one type of boundary control over the other. Thus, equally distributing the probability to enforce each boundary control increases the general capability of the swarm to quickly solve a diverse set of problems.

III. Multiobjective Particle Swarm Optimization

The SOPSO is able to fully automate and improve the speed of single objective optimization with respect to traditional point design methodology. However, in many engineering applications, multiple objectives exist. In most cases, these objectives are in competition with one another, meaning that an improvement in one objective will result in the degradation of another. Thus, no clear single best solution exists. Instead, a set of solutions exists that represents the optimal tradeoff among all of the objectives. This set is commonly known as the Pareto front. An example of a Pareto front may be seen in Figure 2. In this example, two competing objectives (see Eq. (3)-(5)) are simultaneously minimized. A parametric sweep was performed to show all possible J_1-J_2 combinations depicted in blue in Figure 2. The Pareto front for this objective space is shown in red. As can be seen, in order to decrease one of the objectives, the other objective must increase due to the competitive nature of the objectives. This illustrates the notion that all solutions on the Pareto front are known as non-dominated since each solution is equally good. All of the other blue solutions are dominated solutions since there are solutions (that reside on the Pareto front) that are better (one objective can be reduced without increasing the other objective).

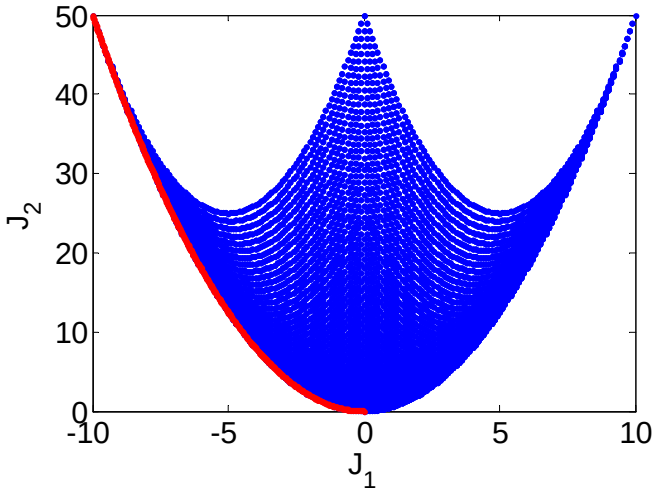


Figure 2: Pareto Front Example

$$J_1 = x + y \tag{3}$$

$$J_2 = x^2 + y^2 \tag{4}$$

$$-5 \le x, y \le 5 \tag{5}$$

The SOPSO algorithm was expanded to efficiently locate and explore the Pareto front for multiobjective optimization problems. The three distinct components of the velocity equation, the inertia, cognitive, and social terms, remain. It is important to note that the inertia component of the velocity equation remains unchanged, identifying the importance of the global and local searching of the algorithm regardless of the number of objectives. However, modifications must be made to determine when to geometrically decrease the value of the inertia and is discussed later. Both the social and cognitive components of the velocity equation must also be altered from the SOPSO algorithm when in the presence of multiple objectives.

A. New Velocity Updates

In the SOPSO algorithm, the velocity was comprised of three components. Two of these components corresponded to the tendency of the particle to return to the best solution it has seen and the tendency of the particle to travel to where its neighbor has seen to be best. Since the concept of best does not necessarily exist for multiobjective problems in which the objectives are in competition, these components to the velocity equation must be modified. In MOPSO, the cognitive position is updated each time an individual locates a new non-dominated solution. This allows the individual to freely travel along the Pareto front. The social component of the velocity equation must also be modified since the quantification of a best neighbor may be impossible in the presence of competing objectives. The social component of the velocity originates from the external archive.[5]

B. The External Archive

The social component of the velocity equation essentially forms the communicative ability of the algorithm, permitting interaction of various individuals. However, due to the presence of competing objectives, the social component must also be modified. While the SOPSO is used to locate a global optimum, the MOPSO is used to identify the entire Pareto front. In order to assist in the exploration of the Pareto front, an external archive is used. This archive is limited in size (specified by the designer) and is used to represent the most updated Pareto front the swarm has identified during the optimization. This archive is updated as individuals identify new non-dominated solutions during the optimization process. It is important to note that the archive is limited in size and is not allowed to grow indefinitely. This is necessary in order to encourage the swarm to expand the Pareto front. As new non-dominated solutions are found, these solutions are added to the archive. If the archive is full, the new non-dominated solution replaces a member of the external archive. The member ejected from the archive is determined by the crowdedness distance. The crowdedness distance is computed for each member of the archive and represents the density of the archive in the region about each member. Consequently, high crowdedness distances correspond to less dense regions of the Pareto front. Thus, the member with the lowest crowdedness distance is ejected from the archive when the archive is full.

The archive and crowdedness distance also serves the purpose to identify regions in the design space associated with the social parameter of the velocity equation. In order to determine the design space position for the social component of the velocity equation, a pool is formed from the members of the archive with the highest crowdedness distances. A member from this pool is randomly selected for each individual in the population and serves as the location in the design space used in the social component of the velocity equation. In two objective optimization problems, the Pareto front is a curve confined to a plane in objective space (although it may be discontinuous). In this case, members of the archive that are at the extremes of the Pareto front are assigned infinite crowdedness distances. This ensures that the endpoints of the archive are always present in the pool used for the social parameter in order to promote exploration and expansion of the edges of the Pareto front. Essentially, the social component provides a tendency for each individual to travel and explore less dense regions of the Pareto front. When more than two objectives are used, the Pareto front becomes a surface or hypersurface in which calculation of the “edge” of the Pareto front becomes prohibitively challenging and was not computed. In summary, the careful selection of ejected members from the archive and the exploration of less dense regions of the Pareto front allow the MOPSO algorithm to continuously push the archive to the true Pareto front and expand to identify the entire Pareto front.

IV. Convergence and Termination

The convergence, or termination, criterion for the SOPSO is governed by the ability of the swarm to locate better solutions. The SOPSO algorithm continues to iterate until convergence is satisfied or the maximum number of iterations has been performed, as specified by the user. Convergence of the solution is determined by integrating the best solution found over a given number of most recent iterations specified by the user. If this integral is within a given percentage of the most optimal solution of the most recent iteration, then the algorithm terminates.

Unlike the SOPSO, the MOPSO is attempting to locate the front of Pareto optimal solutions. In general, this front consists of an infinite number of points, and the points in the archive that are describing the front are constantly changing. Thus, the points in the archive, although located in the near proximity of the true Pareto front after a sufficient number of iterations, are not necessarily asymptotically converging to the true Pareto front. Thus, an alternative criterion must be utilized to determine when the swarm's Pareto front is in proximity of the true Pareto front and when sufficient diversity exists in the front. Both goals can be accomplished by monitoring the recent history of the crowdedness distance. When the crowdedness distance remains fairly static for a sufficient number of iterations, the swarm front will have most likely converged to the global Pareto front with sufficient diversity that spreads the points in the archive throughout the entire Pareto front. Thus, the history of the crowdedness distance is integrated over a number of most recent iterations specified by the user. If this integral is less than a tolerance specified by the user, then the MOPSO algorithm terminates.

V. Additional Capabilities of SOPSO and MOPSO

In many engineering applications, the designer is interested in finding the best solution (in the case of SOPSO) or set of non-dominated solutions (in the case of MOPSO) subject to a number of constraints. In the case of single objective problems, enforcing constraints via penalty functions is quite popular for its ease of implementation. The penalty function approach assigns poor objective values to those solutions that violate any constraints. The magnitude by which to penalize these functions is problem dependent. Also, penalizing infeasible solutions theoretically allows the algorithm to explore infeasible areas of the design space. If the penalty function is designed improperly, the algorithm may return what appears to be an optimal solution but is, in reality, infeasible. Both the SOPSO and MOPSO have the capability to perform constrained optimization but do not use penalty functions. The algorithms simply ignore infeasible solutions. Thus, these solutions will not be remembered by the individuals in the swarm as they search the design space. Consequently, infeasible locations of the design space cannot serve in the cognitive and social components of the velocity equation. When the SOPSO and MOPSO convert to local searching algorithms (with a low inertia), the individuals will not return to these infeasible solutions.

A major advantage of SOPSO and MOPSO is the parallel nature of the optimization process. At any given iteration, each individual in the swarm must be evaluated in order to obtain all of the objective values. The evaluation of an individual is simply a function call to the "black-box" used to model the nature of the problem. Hence, each individual in the swarm can be evaluated simultaneously. This lends to the nature of parallel computing in which a cluster could be used to rapidly increase the execution speed of each algorithm. Each individual in the swarm could be evaluated simultaneously on different nodes of a cluster, an advantage that cannot be utilized by serial methods such as gradient algorithms.

VI. Comparison and Validation

Due to the numerical nature of each algorithm mentioned in this report, it is impossible to know for certain that the global optimum has been found. The global optimum can be located and proven for analytical problems only when using methods such as the calculus of variations. However, the intent is to implement these algorithms to non-analytical trajectory problems. Thus, several analytical and non-analytical test cases were used to gain confidence in each algorithm's capability to locate the global optimum or Pareto front.

A. Analytical SOPSO Validation

Initially, a relatively simple analytical function was used to determine the capability of the SOPSO to locate the global optimum. The intent is to maximize the objective function J. In order to have a trade space that does not expand infinitely in certain directions, the design variables (x and y) are constrained to $-20 \le x,y \le 20$. The objective function J is a combination of an inverted paraboloid, a sloping plane, and a sinusoidal function and is expressed in Eq. (6). A surface plot of the objective function may be found in Figure 3. As can be seen in Figure 3, the optimal value is highly dependent on the initial guess if a gradient algorithm is used. In many cases, the designer does not have the luxury of supplying a good initial guess. This is due to the fact that the designer does not have the capability of performing a parametric sweep of the design space (used to create Figure 3) when the evaluation of each test solution becomes prohibitively time consuming.

$$J = -x^2 - y^2 + 2x + y + 100\sin(x) + 200\sin(y) \tag{6}$$

As mentioned before, the swarm in the SOPSO moves. Individuals in the swarm communicate and retain their knowledge of the best solution found. This, in conjunction with the reduction in inertia, ensures that the swarm converges on the local (and hopefully global) optimum. Initially, the individuals in the swarm are randomly dispersed throughout the entire design space. This ensures that the swarm has a diverse set of solutions in order to ensure that no region of the design space is given preference. Various SOPSO optimizations were performed by changing the random initial guess. The SOPSO algorithm was able to converge on the global optimum regardless of the initial guess. This is an important quality as the designer usually does not have a good initial guess. Figure 4 illustrates the global and eventual local searching methods of the SOPSO through contours of Eq. (6). As can be seen, during the global search, the individuals, represented as dots, are well spaced throughout the entire design space. This is due to the high inertia early in the optimization process, encouraging each particle to be independent in its search. Near the end of the optimization process, the particles converge and communicate with low values of inertia. As each individual remembers its best solution and communicates with other individuals in the swarm, the swarm collapses on the global optimum. During the final iteration (not shown), all of the individuals collapse into the single point corresponding to the global optimum.

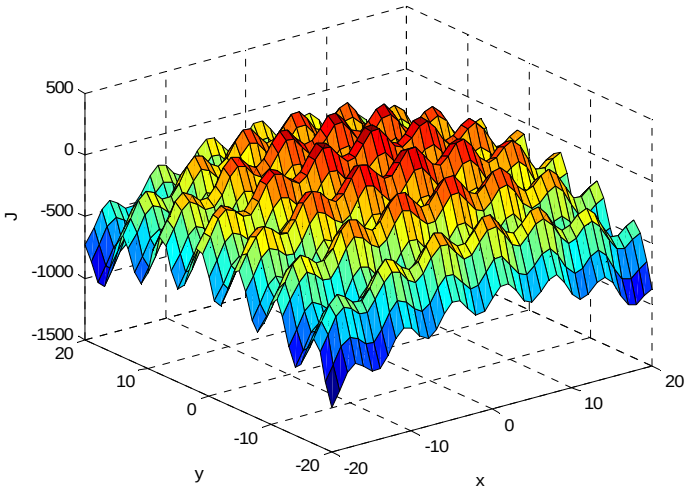
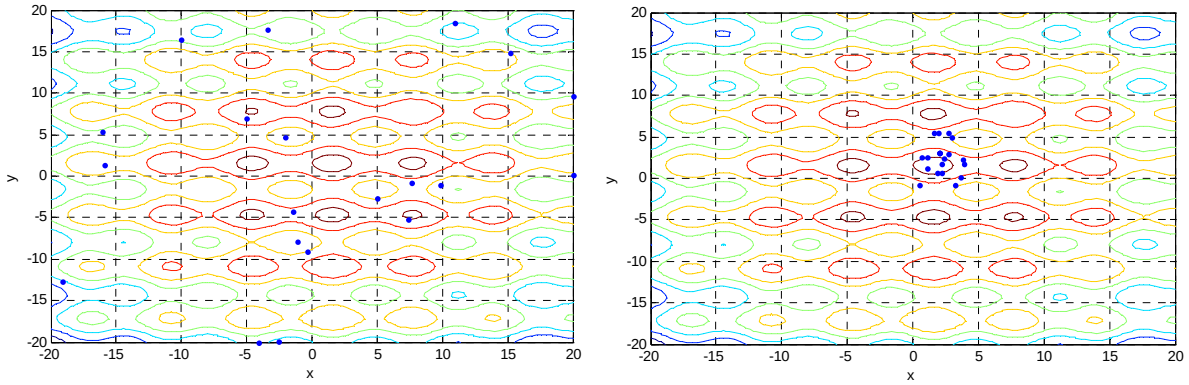


Figure 3: Simple Test Case



a.) Global Search **b.) Local Search**

Figure 4: Snapshots of the SOPSO Algorithm

B. Analytical MOPSO Validation

Three published test cases were used to validate the multiobjective particle swarm.[5] The first test case consists of a two-objective unconstrained optimization. The second test case is a two-objective constrained optimization (note that the objectives in the second test case are not the same as the first test case). The final test case consists of an unconstrained three-objective optimization. In all three optimization problems, the goal is to minimize the objectives. Each objective is denoted as J_i where i is the objective index. The design space for all of the problems is two-dimensional in x and y.

The Pareto front for a two-objective, unconstrained optimization (MOP6), Eq. (7) - Eq. (9), was generated using the MOPSO. The results were then compared to the published data (Figure 5). As can be seen from Figure 5, excellent agreement exists between the published results and the MOPSO algorithm. Note that the Pareto front is discontinuous. This information is very important since it allows the designer to visualize the instances in which improving one objective would result in a dramatic degradation of the other objective as the solution moves from

one branch of the Pareto front to the other. The Pareto front for a two-objective, constrained optimization (MOPC1), Eq. (10) - Eq. (12), was also generated using the MOPSO. The results were once again compared to the published data (Figure 6). As can be seen from Figure 6, excellent agreement exists between the published results and MOPSO results. This served to build confidence in the ability of the MOPSO algorithm to identify Pareto fronts in a constrained environment.

MOP6:

$$J_1 = x \tag{7}$$

$$J_2 = (1+10y)*\left[1-\left(\frac{x}{1+10y}\right)^2 - \frac{x}{1+10y}*\sin(8\pi x)\right] \tag{8}$$

$$0 \le x, y \le 1 \tag{9}$$

MOPC1:

$$J_1 = 4x^2 + 4y^2 \tag{10}$$

$$J_2 = (x-5)^2 + (y-5)^2 \tag{11}$$

$$0 \le x \le 5,\ 0 \le y \le 3 \tag{12}$$

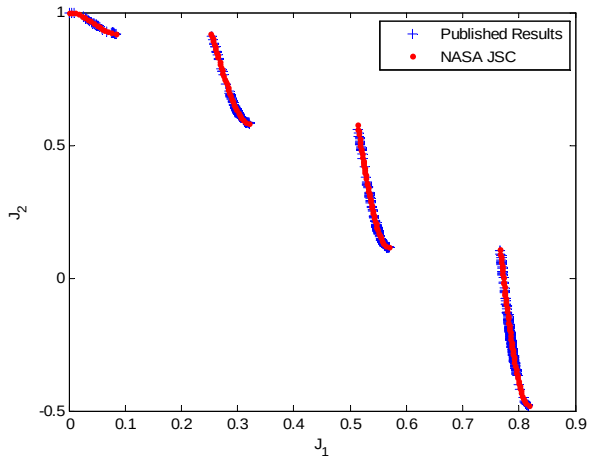


Figure 5: MOP6 Comparison

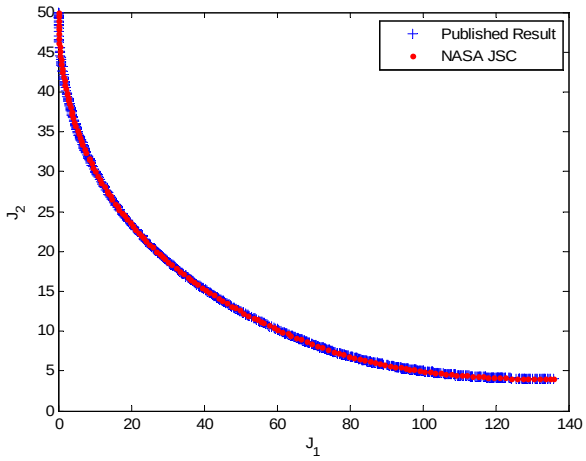


Figure 6: MOPC1 Comparison

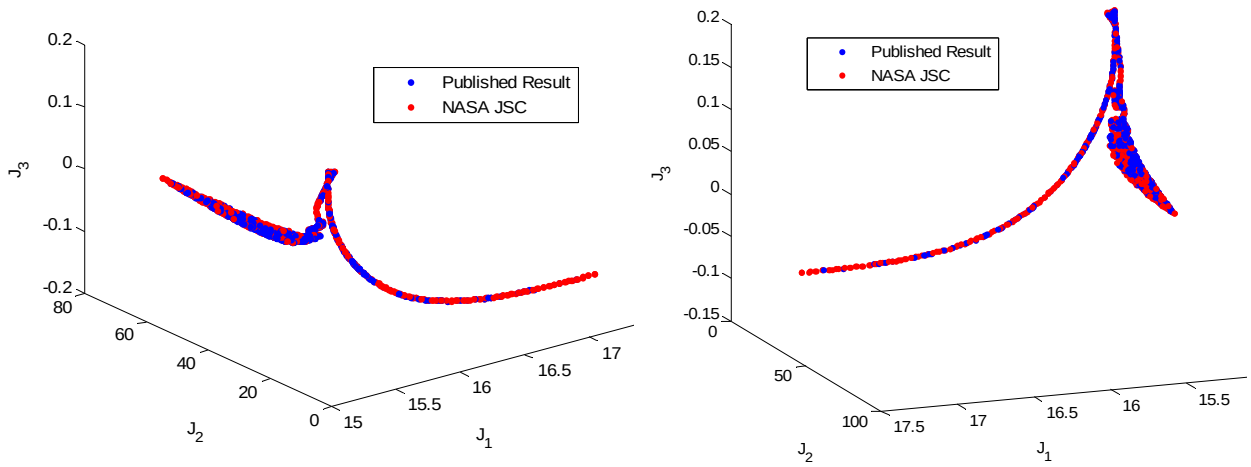
The Pareto front for a three-objective, unconstrained optimization (MOP5), Eq. (13) - Eq. (16), was constructed using MOPSO. The results were compared to the published data (see Figure 7). As can be seen, excellent agreement exists between the published results and the MOPSO. Note that in a three-objective optimization problem, a Pareto surface may form. In this case, both the published results and MOPSO identified the surface. Note that an apparent planar curve extends from the upper portion of the Pareto surface. Both algorithms identified this curve. However, the MOPSO is able to identify a more diverse set of Pareto solutions along this line. Also note that both algorithms identified the surface with a sufficient number of points. The apparent gaps in the Pareto surface are only due to the overlapping in plotting with the MOPSO and published results.

$$J_1 = 0.5*(x^2 + y^2) + \sin(x^2 + y^2) \tag{13}$$

$$J_2 = (3x - 2y + 4)^2/8 + (x - y + 1)^2/27 + 15 \tag{14}$$

$$J_3 = 1/(x^2 + y^2 + 1) - 1.1e^{(-x^2-y^2)} \tag{15}$$

$$-30 \le x, y \le 30 \tag{16}$$



a.) View 1 **b.) View 2**

Figure 7: MOP5 Comparison

C. SOPSO Mars Entry Trajectory Validation

The ultimate goal of developing the SOPSO and MOPSO algorithms is for application to MSL entry trajectory optimization problems. In order to do so, the function evaluations are no longer analytical and require the computationally expensive execution of the Program to Optimize Simulated Trajectories (POST). The algorithms interface with POST using the 200-node cluster system of the Exploration Systems Engineering Branch (ESEB) at LaRC, capitalizing on the parallel nature of the algorithms. The single-objective particle swarm algorithm was compared to Isight used by members of the ESEB at LaRC. In these single-objective optimization cases, open-loop MSL entry simulations were performed using POST. Both an unconstrained and constrained optimization problem was performed for the MSL 06-01 point design. In both optimizations, the geodetic parachute deployment altitude (gdalt) was used as the objective function and was maximized. The design variables include a constant early bank angle, a constant late bank angle, and a velocity switch that governed the instantaneous switch from the early bank to the late bank. The results for the unconstrained optimization may be found in Table 1. As can be seen, the objective function is identical using both algorithms with small variations in the bank profile. Thus, a family of optimal solutions exists in this region.

An open-loop constrained optimization was also performed to determine how well PSO and Isight compare for difficult problems. The imposed constraints were the following: 1.) maximum g-loading less than 15 Earth g's, 2.) maximum heat rate less than 100 W/cm^2, and 3.) total heat load less than 3000 J/cm^2. The objective function remained as gdalt, and the design variables also include the entry flight path angle. The results may be found in Table 2. As can be seen, the PSO was capable of finding a marginally better solution. However, even with comparable objective values, the PSO and Isight converged to very different locations within the design space. This may be largely due to the apparent relation between entry flight path angle and early bank angle. A steep entry flight path angle with a smaller early bank angle and a shallow entry flight path angle with a larger early bank angle may result in a similar trajectory at the end of entry and would result in similar performance while meeting all of the constraints.

Table 1: Comparison of PSO and Isight (unconstrained)

	PSO	Isight
gdalt (km)	12.398	12.398
Switch (m/s)	4363.84	4363.84
Early Bank (deg)	179.825	180
Late Bank (deg)	0.1784	0

Table 2: Comparison of PSO and Isight (constrained)

	PSO	Isight
gdalt (km)	12.458	12.456
Switch (m/s)	4280.02	4473.067
Early Bank (deg)	112.22	159.43
Late Bank (deg)	0.27	0
Flight Path Angle (deg)	-16.5147	-15.9817

VII. Application to MSL Reference Trajectory Design

A. Reference Trajectory Overview

Throughout the hypersonic phase of entry, MSL will use a modified Apollo guidance to perform the first precision landing on Mars by controlling the supersonic parachute deployment to within 10 km of the target. Throughout entry, the modified Apollo guidance compares the vehicle's current drag acceleration, altitude rate, and range to a reference trajectory. The vehicle's perturbation from the reference trajectory is used by the Apollo-derived Entry Terminal Point Controller to determine the necessary change in vertical lift via bank modulation to converge the range flown to the target.[1] The design of the reference trajectory is the responsibility of NASA JSC. As mentioned before, the traditional point design process used by the MSL team is manual and labor intensive. During the point design process, the designer would design the reference trajectory in an attempt to maximize the supersonic parachute deployment altitude while minimizing the miss distance. Each candidate reference trajectory was used in three dispersed scenarios that are used to approximate the worst-on-worst dispersions for altitude and miss distance observed in a full Monte Carlo.

B. Two-Objective MOPSO

The MOPSO algorithm was applied to the MSL 05-22 point design to determine if greater insight to the design process could be provided in an automated manner. The MOPSO was limited to vary only the early and late bank angles and the entry flight path angle. This limitation was imposed to reduce the dimension of the design space to be consistent with the limitations of the manual, point design process in order to make fair comparisons. A comparison in the Pareto front generated by MOPSO to the manual 05-22 point design may be seen in Figure 8. It appears that the 05-22 point design is dominated by the Pareto front generated by MOPSO. This may be easily seen in the 400 km supersonic parachute deployment altitude gain that could be achieved while maintaining the same range error ellipse length or in the 3 km reduction in range error ellipse length that could be achieved for the same supersonic parachute deployment altitude. However, it is important to note that other considerations, such as control saturation and g-loading, were accounted for when manually designing the 05-22 point design reference trajectory that were not included when using MOPSO since these other considerations were not directly quantified in the point-design process. When these other considerations were taken into account, it was not possible to quantify their impact on the performance of the vehicle when using the traditional point-design process. However, the performance degradation is completely quantified by observing the Pareto front. It is important to note that these other considerations were only qualitatively accounted for without providing any quantitative, documented criteria. If other considerations are important, then these considerations can be included as other objectives. MOPSO could then be used again to fully quantify the true optimal trade among all objectives and metrics of concern to the designer.

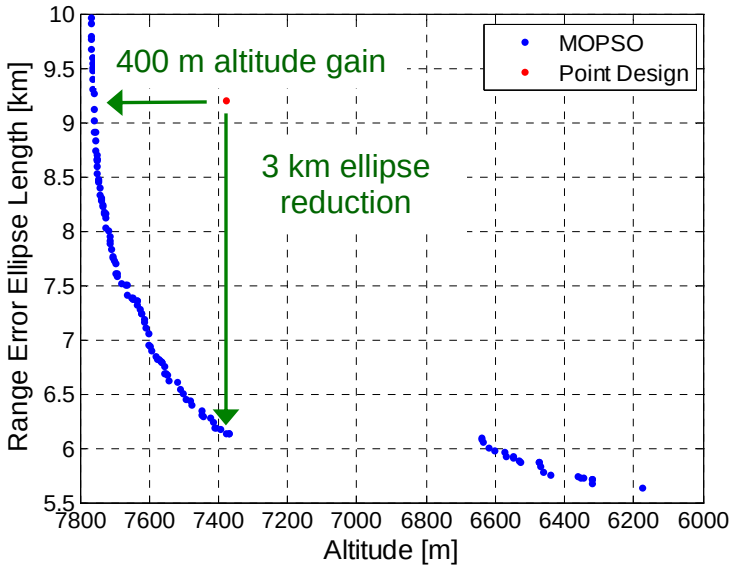


Figure 8: 05-22 Comparison

While the Pareto front itself provides the set of non-dominated solutions that can be used to select the optimal design in the presence of competing objectives, the investigation of the structure of the Pareto front also serves a useful purpose. As can easily be seen in Figure 8, a significant gap appears between two branches of the Pareto front. In order for MOPSO to reduce the range error ellipse length to about 6 km or below, a major reduction in supersonic parachute deployment altitude is required. The trajectories of the two points at the edges of the gap were further analyzed, and it was clear that the two trajectories mainly varied in entry flight path angle. In order to reduce the range error ellipse length, the entry flight path angle was significantly shallowed from -15.25 deg (left gap trajectory) to -14.5 deg (right gap trajectory). Hence, a shallow entry trajectory results in the vehicle flying at

higher altitudes longer. Near the end of the trajectory, the vehicle must dive into the atmosphere due to passage through the equilibrium glide boundary. This diving effect results in a smaller range error ellipse length in the presence of dispersions at the consequence of a major reduction in supersonic parachute deployment altitude.

C. Three-Objective MOPSO

In order to further investigate the capability of MOPSO, a third objective was added, g-loading, and was minimized. Since all three objectives are in competition with each other, a Pareto surface is formed (see Figure 9). The three-dimensional Pareto surface was projected into the altitude and range error ellipse length plane and compared to the two-dimensional Pareto front (see Figure 10). As can be seen, MOPSO was able to capture the original Pareto front even with the added objective.

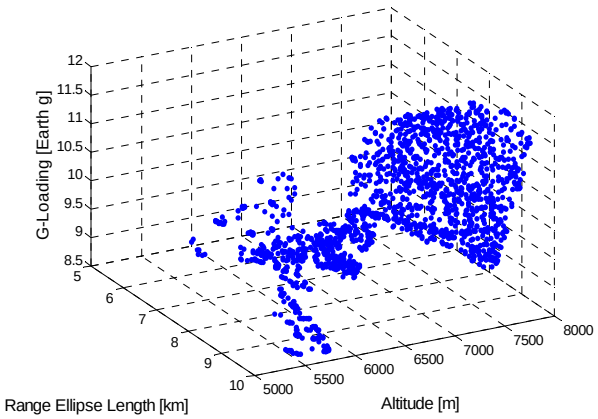


Figure 9: 05-22 Pareto Surface

It is rather difficult to gain immediate insight into the Pareto surface from the three-dimensional view. However, the Pareto surface generated by MOPSO can provide useful insight when various projections are viewed. The same 2D projection shown in Figure 10 is color coded according to entry flight path angle (see Figure 11). As can be seen, the range error ellipse length and altitude are divided into sections that are defined by the entry flight path angle. High altitudes may only be achieved with steep entry flight path angles, as expected. However, the influence of entry flight path angle can now be quantified and directly correlated with the objectives.

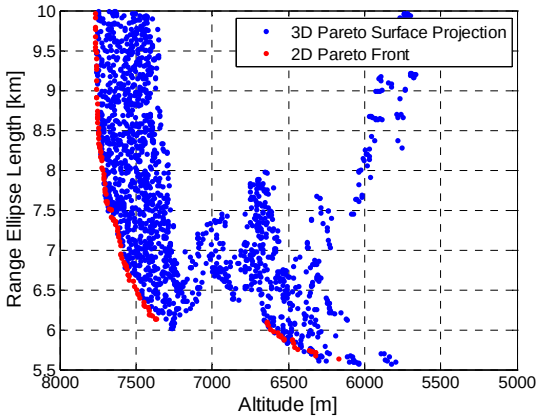
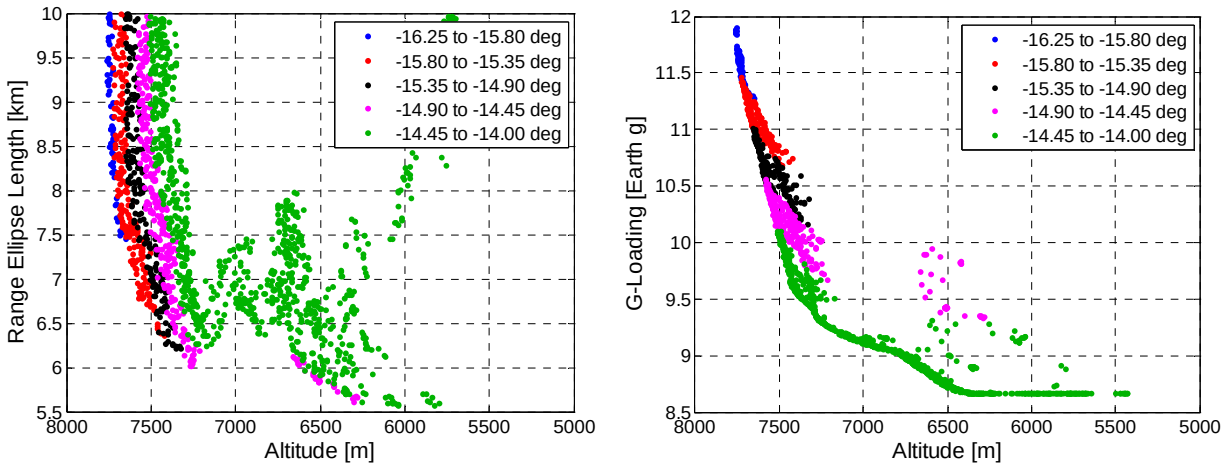


Figure 10: 2D and 3D Pareto Optimal Solutions Comparison

As can be seen in Figure 11, in order to achieve higher altitudes, the g-loading on the vehicle must also increase. As the entry flight path angle is steepened to achieve higher altitudes, the vehicle will experience a higher g-loading. Consequently, the impacts of trajectory performance can be directly related to vehicle design. The vehicle's structural tolerance may result in limitations in g-loading and, consequently, trajectory performance. The inherent multidisciplinary coupling of trajectory and vehicle design can be assessed in a visual environment. Vehicle constraints, such a g-loading, can be converted to be an objective in order to view the sensitivity to other objectives, such as trajectory performance (as seen in Figure 11). Note that the lower portion of the graph is flat, resulting in relative constant g-loading. This is due to the value of the shallow bound of the entry flight path angle.



a.) Range Ellipse Length and Altitude Projection **b.) G-Loading and Altitude Projection**

Figure 11: 2D Projections with EFPA Coloring

VIII. Conclusion

An attempt to improve upon the traditional point design method for reference trajectory design was explored through the construction of a single and multiobjective particle swarm optimization algorithms. Both algorithms provide a way to perform relatively fast and automated optimization that was originally performed in a time consuming, manual way. The MOPSO algorithm was able to construct Pareto fronts in a few hours that provided quick insight into entry trajectory design characteristics that took years to understand through the traditional point design process. The automated construction of the Pareto front allows the designer to focus attention on only the set of best solutions. The optimal trade associated with conflicting objectives was also quantified and visualized, allowing the designer to make intelligent decisions in the design of the reference trajectory. This work serves as a beginning of a fundamental departure from the traditional point design process. It is important to note that the environment necessary to run MOPSO for MSL reference trajectory design was completed without providing sufficient time to further explore the capability of MOPSO for reference trajectory design. Consequently, the capability of MOSO is not simply limited to the results shown. More elaborate reference trajectory bank profiles and the addition of objectives such as peak heat rate could be included in the future to develop the optimal trade in all objectives of interest to the MSL team. While great strides were made to improve on the speed and quality of reference trajectory design, much improvement could be made. Both the SOPSO and MOPSO algorithms require computationally expensive POST runs to be performed. While these algorithms attempt to identify optimal solutions and Pareto fronts with a minimum number of POST runs, surrogate modeling may provide a way to dramatically increase execution speed through highly accurate functional approximations to the output of POST. Hence, surrogate modeling may provide the ability to generate n-dimensional Pareto fronts and apply other advanced MDO techniques nearly instantaneously, resulting in a major departure from the traditional point design process.

IX. Acknowledgments

The author would like to thank Gavin Mendeck at the NASA Johnson Space Center for his insight and teachings as an excellent Co-op mentor. The author would also like to thank those at the NASA Langley Research Center: John Aguirre, Jody Fisher, Rob Maddock, David Way, and Loreyna Yeung for their assistance in using the cluster and optimization comparisons. Special thanks to Leticia Cagnina for providing his data for comparison of the analytical MOPSO test cases.

References

[1]Mendeck, G. F., and Carman, G. L., "Guidance Design for Mars Smart Landers Using The Entry Terminal Point Controller," AIAA-2002-4502, *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Monterey, California, 5 Aug – 8 Aug 2002.

[2]Kennedy, J., and Eberhart, R., "Particle Swarm Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, IEEE, Perth, WA, Australia, 1995, pp. 1942-1948.

[3]Shi, Y., and Eberhart, R., "Empirical Study of Particle Swarm Optimization," *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 3, IEEE, Washington, DC, 6 Jul – 9 Jul 1999.

[4]Venter, G., and Sobieszczanski-Sobieski, J., "Particle Swarm Optimization," *AIAA Journal*, Vol. 41, No. 8, Aug 2003, pp. 1583-1589.

[5]Cagnina, L., Equivel, S., and Coello Coello, C. A., "A Particle Swarm Optimizer for Multi-Objective Optimization," *Journal of Computer Science and Technology*, Vol. 5, No. 4, Dec 2005, pp. 204-210.