SIMULATED ANNEALING MANEUVER PLANNER FOR CLUSTER FLIGHT

Allen G. Brown[(1)], Dr. Matthew C. Ruschmann[(2)], Dr. Brenton Duffy[(3)], Lucas Ward[(4)], Dr. Sun Hur-Diaz[(5)], Eric Ferguson[(6)] and Shaun M. Stewart[(7)]

Emergent Space Technologies Inc., 6411 Ivy Lane, Suite 303, Greenbelt, MD 20770
(301) 345-1535
[(1)]allen.brown@emergentspace.com
[(2)]matthew.ruschmann@emergentspace.com
[(3)]brenton.duffy@emergentspace.com
[(4)]lucas.ward@emergentspace.com
[(5)]sun.hur-diaz@emergentspace.com
[(6)]Jet Propulsion Laboratory (JPL), 4800 Oak Grove Dr., Pasadena, CA 91109, (818) 354-4321
eric.w.ferguson@jpl.nasa.gov
[(7)]Helios Solutions, LLC, 2101 NASA Pkwy., Houston, Texas 77058, (281) 483-5920
sstewart@heliossolutionsllc.com

Abstract: *A simulated annealing (SA) heuristic search algorithm has been implemented for creating maneuver plans for guidance of a satellite cluster. The algorithm is designed to support a variable number of satellites (also called modules) and multiple maneuver types including: ingress into the cluster, egress from the cluster, station-keeping, cluster reconfiguration, cluster defensive scatter, and regathering the cluster after scattering. The maneuver planner searches over a discretized parameter space varying the earliest and latest maneuver times and target relative orbit elements (ROEs) while using a lower-level linear programming multiple-burn solver to evaluate optimal maneuvers that satisfy a set of cluster-wide and module-specific constraints and objectives. These algorithms allow for management of competing operational objectives and constraints while minimizing module ΔV and the probability of inter-module collision. This paper presents an overview of these maneuver planning algorithms and the message-based software service that encapsulates them for supporting cluster flight.*

Keywords: *cluster flight, simulated annealing, maneuver planning, software service, System F6*

1. Introduction

This paper presents an overview of the algorithms and design of an on-orbit, message-based, software service application that produces orbital maneuver plans for a cluster of satellites, also called cluster modules, while managing module consumables and operational risks within the cluster. These algorithms and software could be used to generate and update maneuver plans required to support the full mission lifecycle of a spacecraft cluster and operate with a variable number of modules. The problem background and driving requirements of the cluster maneuver planning and software application environments are also presented. The Maneuver Planning Service (MPS) software application uses a Simulated Annealing (SA) heuristic search algorithm for maneuver planning. This paper describes how SA and its supporting algorithms are used to accommodate cluster flight mission needs to produce operationally useful maneuver plans. The software architecture and high-level design are described along with commentary on the different environments in which the software has been deployed during development. Selected maneuver

planning results are included to illustrate current capabilities along with a discussion on current and future applicability to different domains and potential enhancements.

2. Background

2.1. Cluster Flight Maneuver Planning Needs

In recent years, the Defense Advanced Research Projects Agency (DARPA) has sought to push boundaries of satellite system architecture via System F6, a program that envisioned a cluster of smaller and cheaper satellites that could fulfill the mission of larger and more expensive monolithic satellites. [1, 2]. In a collaborative satellite cluster, the orbit trajectories of the satellites, or modules, would have to maintain a relative line-of-sight between modules that is sufficient to support wireless communication. The probability of collision between modules (Pc) would also have to be controlled to meet mission objectives. The management of consumables for individual modules across the cluster would be a driver for maximizing mission lifetime. Cluster flight, unlike formation-flying mission concepts, has no specific requirements to maintain a precise relative geometry between orbiting modules. Key cluster flight mission concepts are that the number and capabilities of the cluster modules may not be fully determined before the mission and would likely change over the cluster's mission lifetime.

One of the System F6 principal technical components is the Cluster Flight Application (CFA) which administers the flight trajectories of each of the cluster modules [3]. The CFA is software that would fly on one or more of the cluster modules and it has the responsibility for all elements related to trajectory guidance, navigation, and control. This includes planning (creating) and re-planning (modifying) maneuvers for each module within maneuver plans to fulfill the mission objectives. The CFA sends the maneuvers from these plans to the designated modules for the modules to execute. CFA monitors the maneuver execution and develops navigation estimates for the modules in order to exercise closed-loop control. One distinction is that the CFA does not oversee the attitude determination and control or thruster operation of a module bus. Instead, a dedicated CFA service acts as a module interface and interacts with the bus systems for each maneuver execution. The module systems have the responsibility for executing the necessary actions to meet the thrust magnitude, direction, and timing prescribed in the burn. The CFA maintains a description of each module's capabilities to ensure that maneuvers are operationally feasible.

The CFA nominally shepherds individual modules along station-keeping trajectories to fulfill mission requirements. The two principal goals of station-keeping are to minimize the delta-V over the mission lifetime and to maintain a low Pc for the entire cluster. However, over the lifetime of a cluster other types of maneuver plans are required. Individual modules need to ingress (enter) an existing cluster in order to grow the cluster or to replace other modules. Likewise modules will need to egress (leave) a cluster for module end-of-life scenarios or to transfer to other clusters. It may be necessary to adjust the configuration of the cluster (cluster reconfiguration) for many reasons including growing the size of the cluster to host a future ingressing module. As with station-keeping, these kinds of maneuver plans must balance delta-V use across the cluster while maintaining a low Pc and fulfilling other mission needs. A former objective of System F6, which the cluster flight algorithms and software in this paper supported,

was to execute a defensive scatter of the cluster modules in order to avoid a potential threat. Each module was required to be at least 10 km from the original orbit trajectory of any cluster module 5 minutes after the reception of a scatter command from the ground. After a scatter occurred, the CFA would subsequently regather the cluster modules back into a new mission configuration without ground operator intervention [2]. Any of these maneuver planning problems may require planning horizons of up to several orbit periods in the future to take advantage of multiple revolution transfers to the desired target orbits.

Given that the cluster size can vary widely, it is desirable that ground interactions and corrections to CFA do not necessarily scale with the number of modules in the cluster. Module trajectory maneuver planning nominally proceeds without requiring ground interaction. This, in turn, alleviates much of the burden placed on ground operators when adding another module to the cluster.

2.2. Cluster Flight Application Architecture Drivers

The CFA is architected for on-orbit execution as a distributed set of message-based software services [3]. These CFA component services are connected by an underlying message bus that operates over the wireless communication links between the cluster modules. A service-based architecture enables deployment flexibility for placing component services on different modules and provides system redundancy via component fail-over. A message-based interface also allows deployment flexibility across different platforms having varying capabilities. For example, a service could be deployed on, or even re-hosted on, different modules over the life of the cluster even if the computing platform differs. Deployment flexibility is especially important to CFA because target computer platforms had not been selected at the start of CFA development. Another option is that with a dedicated ground-space communications link, a service can be deployed on the ground as well as on-orbit. This service-based architecture approach has already supported rapid prototyping, testing, and upgrades during CFA development.

The Maneuver Planning Service (MPS) is the CFA software component that is responsible for creating and updating cluster maneuver plans to satisfy all of the needs of cluster flight. Other CFA services determine the need for a maneuver plan and describe the maneuver planning problem domain in the form of a service request, the maneuver planning request. MPS processes this request and issues a maneuver planning result that contains all of the updated and newly-created maneuver plans for the cluster modules. This request-response interaction allows MPS to remain essentially stateless (with respect to maneuver planning problems) between requests and simplifies MPS complexity. The maneuver planning request specifies all aspects of the maneuver planning problem as well as a time limit for MPS to calculate an answer. The calculation time limit ensures that an answer from MPS will be available to the CFA in time for execution.

Among other information, the maneuver request contains the CFA estimated orbital states of the cluster modules, the descriptions of the modules including their capabilities and limitations, and designations on the desired target orbit, or orbits, for each module that is being maneuvered. There is an upper limit on the total number of modules MPS can accommodate, currently twenty

modules. It is expected that each request will typically vary in the maneuver planning problem type and in the modules being planned. Additionally, MPS does not limit CFA to re-plan all known cluster modules with a maneuver planning request. It is also important to recognize the distinction between planning the maneuvers and the execution of the maneuvers. While the CFA is concerned with both, MPS is only involved in the planning. Information on maneuver execution is provided as part of the maneuver planning request so that MPS can merge current in-progress maneuvers into its future plans. Since MPS is stateless between requests, this allows MPS to freely switch between re-planning current cluster modules for near-term station-keeping execution and future speculative maneuver planning for different purposes such as defensive scatter.

3. Algorithms

3.1. Goals, Limitations, and SA Overview

The overriding goal for the algorithms and software in MPS is to find an operationally useful solution in a constrained amount of processing time. The maneuver planning trade space is quite large in that any request could vary between planning one to twenty modules. Not only can the type of planning request vary (station-keeping, egress, scatter, etc.) but the cluster configuration trade space itself is wide open. Assumptions about the relative module trajectories cannot be assumed in advance so any approach has to be robust. The resulting maneuver plans have to reflect Pc considerations for mission safety, obey operational constraints on the modules' individual propulsion capabilities, as well as meet the desired target orbit(s) and keep delta-V expenditure low to allow for longer module and cluster mission life. As the number of modules increases, many of these can be directly conflicting objectives that involve a very large and rugged solution space. A critical secondary goal is to reduce the overall effort required to prototype, design, implement, optimize, test, and validate MPS algorithms and its software implementation.

The Simulated Annealing (SA) heuristic search algorithm is applied to the maneuver planning problem to search the problem domain search space while a multiple-burn solver (MBS) is used to generate the maneuvers [4, 5]. The MBS utilizes a Linear Programming (LP) solver for computational efficiency [6, 7]. An inertial propagator is used to set up the MBS solution conditions and create the resulting maneuver trajectories for each module from the maneuver plan. Then, the SA algorithm evaluates the resulting maneuvers and trajectories against maneuver planning constraints and objectives. The best search result is reported in the MPS response at or before the processing time limit expires. Figure 1 provides a helpful overview of the MPS processing flow and how the various algorithms in this paper are related.

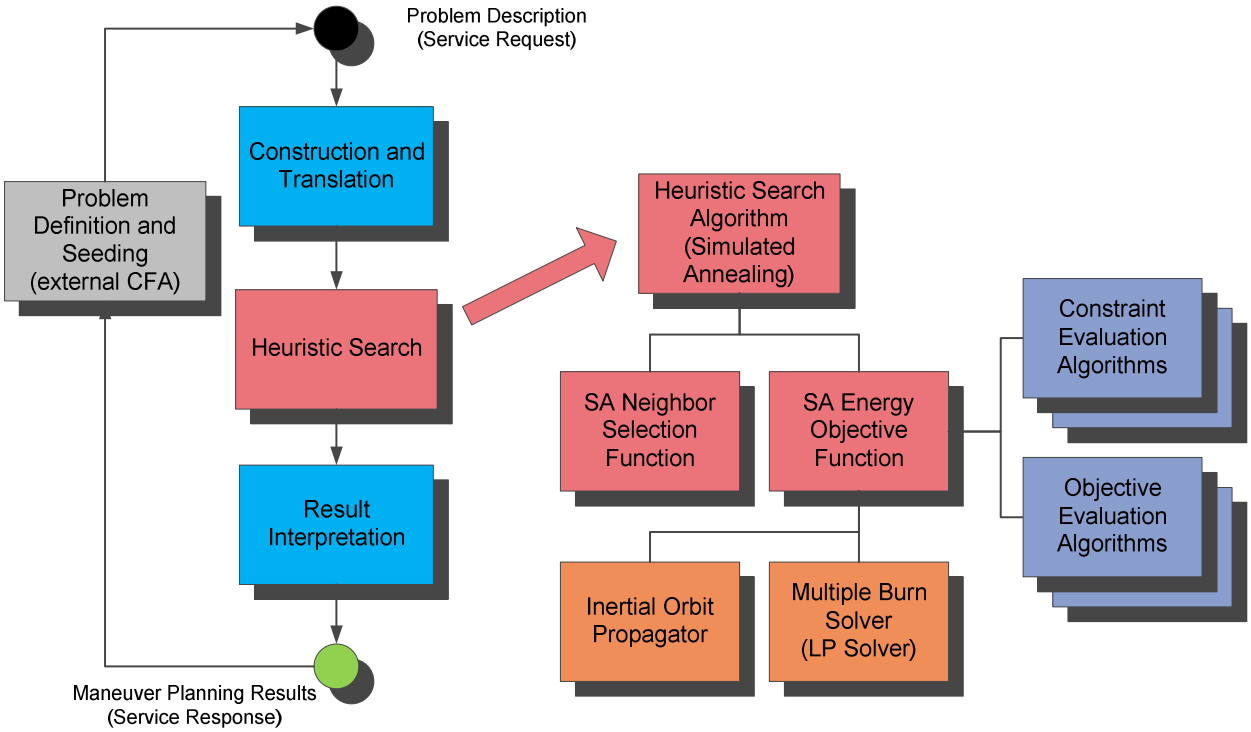

Figure 1. MPS Processing Overview and Supporting Algorithms

SA is a search optimization strategy applied to global optimization problems which can involve large search spaces. Based upon the annealing that occurs with a cooling solid material, a SA search state represents the state of the system undergoing annealing. The variability of this state represents all possible combinations of the problem domain search space. In metallurgy, higher and lower temperatures are related to higher and lower total energy states of the solid's atoms. Similarly, in SA the system search state, *s*, is related to the system energy, *e*, by the problem domain fitness function, *f*, in $e = f(s)$ which is an equivalent to the instantaneous system thermodynamic energy. Similarly, the next system state, *s'*, has an energy, *e'*, via $e' = f(s')$. As the SA search state is changed, the resulting SA energy can change to represent a different fitness where lower energy is a more desirable outcome. The SA search proceeds through the search space by randomly selecting and changing a single element of the SA search state by a random amount. This modification is performed by the neighbor selection function, *N*, via $s' = N(rand, s)$. As demonstrated in [4], randomly changing the state by jumping a far distance across the entire search space leads to poor search performance because a local region is less likely to be explored. Therefore, the neighbor selection function is designed to restrict the worst-case state variation for the problem domain. However, a transition from *s* to *s'* is not always beneficial to the search.

In metallurgy, higher and lower temperatures are used to increase and decrease the system energy states of the solid material as they reach equilibrium. In SA, an equivalent relationship between solution's energy and system temperature is used. The SA temperature, *T*, relates the probability of the search transitions to the annealing cooling of the material. The SA temperature governs the probability of acceptance of a worse solution, *e'* > *e*, via

$$P = exp^{\left(\frac{e-e'}{kT}\right)} \tag{1}$$

where k is similar to Boltzmann's constant and scales T to the expected difference in energy. Equation 1 is always a non-zero probability and is the mechanism to prevent the SA search from becoming stuck at a local minimum. In the classical SA approach, better solutions are always selected while worse solutions might be selected in order to broaden the search space as governed by Eq. 1. Higher temperatures allow worse search regions to be explored and provide the mechanism to escape local minima. Lower temperatures are more likely to reject worse solutions and thus focus the algorithm on local minima. At extremely low temperatures the algorithm resembles a greedy algorithm. The search problem begins with a high initial temperature, *To*, and decreases the temperature as the search progresses on an annealing schedule. Note that SA does not attempt to restrict a unique solution from being re-visited during the search. A well-tuned search lets the system reach equilibrium energy level at each temperature level before cooling thus providing ample opportunity to explore the space at high temperatures while converging upon minima at lower temperatures.

There are several qualitative drivers for the selection of SA for our solution. Given the likely on-orbit computational platform capabilities, it is highly unlikely that an exhaustive solution search is possible so the goal is to find an operationally acceptable solution rather than a provable global optimum. A heuristic search is preferred for overall robustness over a large, variable, and unknown rugged search space. Unlike some heuristic searches, such as genetic algorithms, SA does not require the maintenance of a solution population in memory. Its single state is easily extendable and offers the utility to re-start a search with a prior solution. This allows for faster convergence if the prior maneuver planning problem is similar enough to the current maneuver planning problem. The single SA energy fitness function offers flexibility in combining operational constraints and objectives together. SA is a conceptually straightforward algorithm with many well-known extensions and alternatives allowing our team to focus on many other challenges during development.

3.2. SA Application to Cluster Maneuver Planning

In cluster maneuver planning, not only does the number of modules vary but the number of maneuvers varies widely for reasonably optimal multi-burn transfer trajectory solutions. One direct application of a heuristic search to these types of problems might be to parameterize the maneuvers themselves directly as the search space. However, this makes the search space extremely large and variably-sized. Instead of attempting to embed the module maneuvers into the SA state, we choose to parameterize the SA search space in terms of the well-established orbital transfer two-point boundary problem for each module.

The maneuver planning request contains each module's estimated inertial orbit state and any maneuvers that occur before new maneuvers are created. This is used to define each module's initial inertial orbit position, velocity, and time. Each module's target, or final, inertial orbit state is defined by a cluster reference orbit position, velocity, and time with a set of relative orbit elements (ROEs), which are shown in Fig. 2 [8]. The ROEs comprise a set of state variables [a_e,

x_d, y_d, z_{max}, β, γ] that describe the size, location, orientation, and shape of the module orbit relative to the cluster reference orbit with the module location on the instantaneous relative orbit. These relative state elements are six elements of the SA search space while the initial and final inertial states prescribe the two-point boundary conditions. Each module uses the same cluster reference orbit for maneuver planning while the ROE state elements are searched or held fixed, as designated by the maneuver planning client.

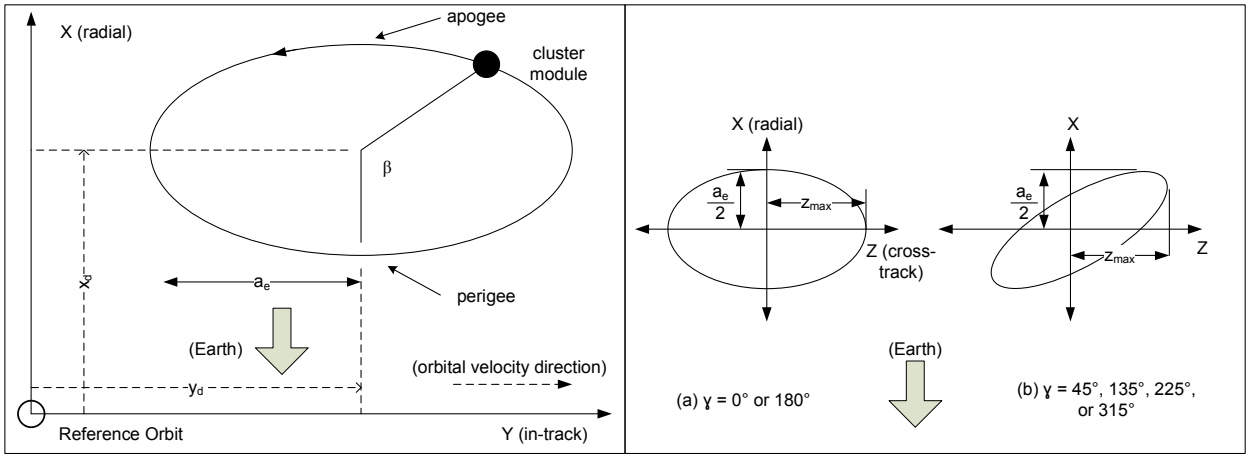

Figure 2. Relative Orbit Elements Used for Target Orbit Definition

The maneuver planning request specifies a time window within which all calculated maneuvers for each module must occur. This maneuver planning window is limited between the times of the modules' initial state estimates and their final target time, as shown in Fig. 3. This window constrains the earliest, *Ti,* and latest, *Tf*, possible burn times which represent two more SA search state elements for each module. The initial orbit state boundary condition is created by propagating each module's initial inertial state forward to the selected *Ti*, including any burns from prior maneuver plans. The final orbit state boundary condition is created by propagating each module's target inertial state, designated by the search state target ROEs and the target orbit time, backwards to the selected *Tf*.

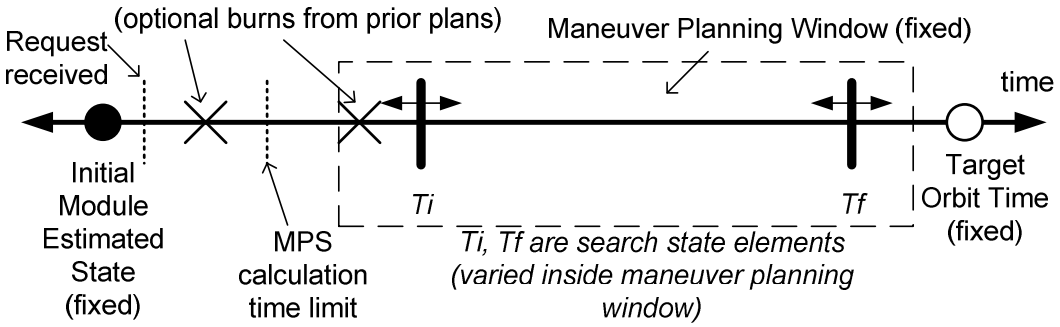

Figure 3. Relevant Maneuver Planning Times for Each Module

Each SA search iteration changes one search state element for a module [*Ti*, *Tf*, a_e, x_d, y_d, z_{max}, β, γ] which results in a new two-point boundary problem. The MBS algorithms, described below,

create a new set of maneuvers and the resulting inertial trajectory from *Ti* to *Tf* which is then evaluated by the SA fitness function.

This parameterization of the search space has multiple benefits. Keeping the maneuvers out of the explicit search space decouples the heuristic search from the algorithms that solve for the maneuvers. This allows various burn solver algorithms to be substituted and tested over the development lifecycle without changes to the SA algorithms. The SA state has a fixed size for each maneuver planning problem depending on the number of modules being planned. Often CFA planning needs do not require variation in ROEs which reduce many maneuver planning requests to only searching on the *Ti* and *Tf* states. When the ROEs are searched they provide additional significant degrees of freedom. While these enlarge the SA search space, the additional degrees of freedom help the SA algorithm minimize delta-V and avoid poor solution regions instead of simply requiring additional burns to meet constraints.

Cluster modules that are not being planned do not increase the search space even if those modules have existing, pre-computed, maneuvers that are scheduled to occur during the planning window. An upper limit to the state size and a software-defined upper limit to the number of modules allow a dedicated subset of the SA state to represent each module in the cluster. This simplifies software implementation and maintenance. Each element in the SA search space is discretized in order to dramatically reduce the number of total solution possibilities in the entire search space. This discretization is configurable for each state element according to the type of maneuver planning problem. Discretization is directly influenced by operational considerations and is one of the keys to finding a practical approach. Other operational considerations on the target orbits are enforced by restricting selected ROE search ranges or by fixing them and thus eliminating them from the search entirely.

The module-specific search states (ROEs, *Ti*, and *Tf*) were a sufficient tool to create useful solutions for all maneuver planning problem types except for cluster scatter. Scatter demands placed an explicit requirement on the resulting trajectory other than the defined *Ti* and *Tf* times and states. Each module's scatter trajectory not only had to have at least 10 km of range from its original trajectory within 5 minutes, it also had to avoid the original trajectories of all of the other modules. In order to avoid a potential threat, it is operationally poor practice to place another module in the same region that another module recently evacuated. As the number of modules increase, or cluster volume decreases, it becomes even more critical for each module to scatter in a "suitable" direction.

The scatter problem is solved by augmenting the SA search space with a supplementary scatter target point in space for each scattering module. This scatter target point is parameterized by azimuth and elevation angles and a range from the initial orbital state propagated to the scatter criteria time of 5 minutes. The module's trajectory is constrained to pass through the scatter target position at the designated scatter criteria time, but the corresponding trajectory velocity is unconstrained. As the SA search alters a target point location, the resulting trajectory and maneuvers change as well. Different angle choices alter the module's initial scatter trajectory to escape the local region and avoid entering restricted regions defined by other modules. Varying the target point range allows the SA algorithm to find higher-energy trajectory solutions that avoid regions where multiple restricted regions may overlap and increase the minimum "safe"

distance for the module to reach. Restricting the target point range search values to 10 km and greater ensure that the SA search and MBS consider trajectories that are more likely to be operationally useful. The combination of varying the scatter target point, selected target ROEs, and *Tf* search elements allow the SA search to find module maneuver plan solutions that integrate the target point constraint and final state with more delta-V efficient strategies than holding either fixed. The separation of SA and MBS algorithms is maintained with the scatter target point representing either an additional internal constraint extension to the two-point boundary problem or the position where two separate two-point boundary problems can be separately calculated. One solution would be dependent upon the other. Figure 4 illustrates the various elements in the SA search state and the role they play in scatter along with an example target point relationship to scatter operational keep-out regions (that exist 5 minutes after the scatter command).

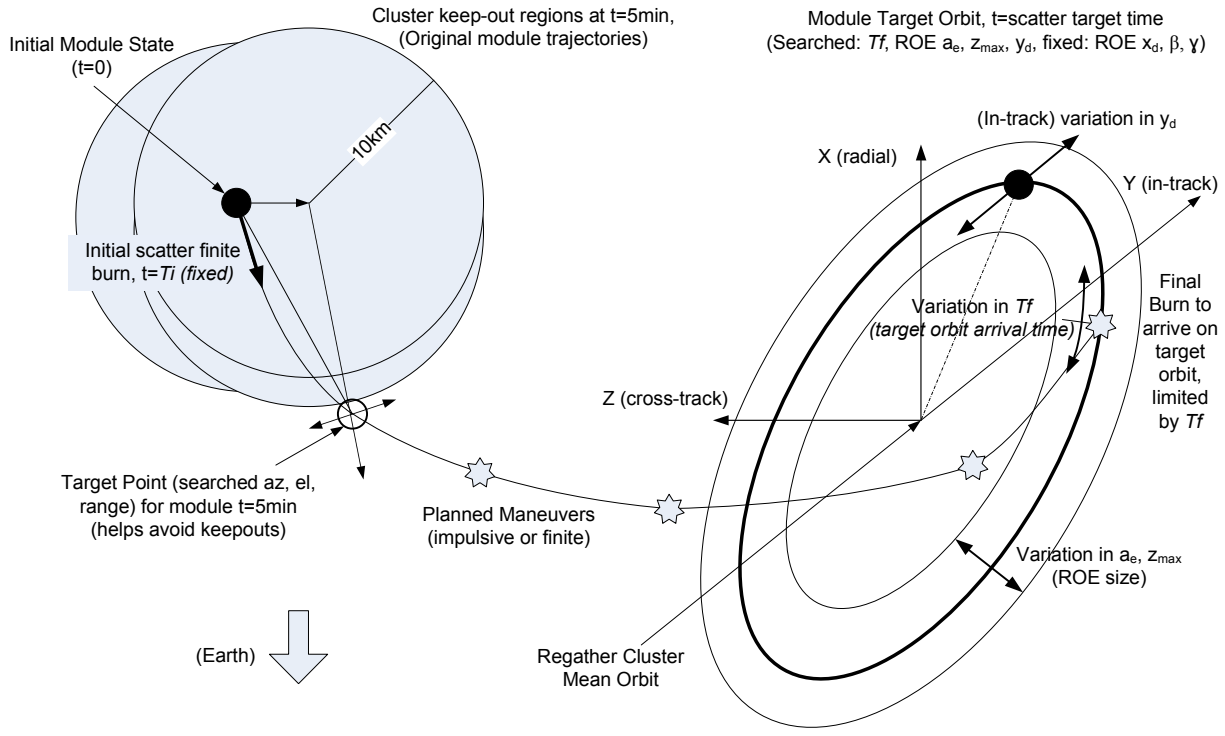

Figure 4. Scatter SA Search Sates and Scatter Keep-Out Constraints.

3.3. SA Constraints and Objectives

It is a straightforward matter to incorporate additional fitness criteria for the SA search metrics because of the single SA energy fitness function. However, there are some fitness criteria which should never be violated in order for the entire maneuver planning solution to be operationally feasible. Other fitness criteria are looser goals that do not necessarily preclude operational use. It is very useful to categorize the various fitness criteria into "constraints", which should not be violated, and "objectives", which have no concept of a violation limit. The constraints are classified with a discontinuous operator so that when violated they add SA energy relative to the amount of violation. Under the violation limit they add no SA energy. Objectives add SA

energy with a continuous operator where $e_{obj} \to 0$ in the ideal case. The constraints are weighted so that a single constraint violation always adds more SA energy than the objectives. This separation and weighting proves to be very useful when searching for a scatter maneuver plan. Scatter planning involves all modules, and as the number of modules increases, many more solutions of the search space violate one or more constraints. With the greater distinction between constraint violations and objectives, the search has the opportunity to explore many solutions having constraint violations with a high T then converge into local minima with no constraint violations as T is decreased. This separation of criteria means that each maneuver planning solution is immediately classified as operationally feasible or infeasible simply by checking for the existence of SA constraint energy contributions. No additional calculations are required. These constraints can be selectively applied or weighted differently to tailor them to the maneuver planning problem.

The primary SA objective is to minimize the delta-V of each module for the calculated maneuver plan. In the maneuver planning request each module is assigned a delta-V limit based on its remaining operational delta-V budget and mission lifetime. Rather than a strict per-module delta-V objective, this objective attempts to minimize the mean of the ratio of delta-V planned for each module compared to its per-module limit. This has the desirable effect of weighting the maneuvers from modules with a lower delta-V limit greater than those with a higher delta-V limit. The resulting behavior is that modules with lower delta-V limits generally use less delta-V than other modules with higher limits. This objective and maneuver planning request delta-V limit is the mechanism by which MPS supports higher-level CFA consumables management. This provides CFA the flexibility to request maneuver plans involving fuel-limited modules and also balance module delta-V use over the cluster mission.

Many operational criteria are successfully implemented as constraints. Each module also contributes a constraint energy penalty if it exceeded its delta-V limit. This enables the solution to find and significantly prefer solutions where all modules are at or under operational delta-V use compared to the objective alone. Additionally, there are several operational constraints on the module trajectories. A minimum inter-module distance (IMD) constraint adds energy whenever two modules came within a prescribed lower range limit. Enforcing minimum IMD constraints support cluster Pc operational objectives. This minimum IMD is applied throughout the maneuver planning windows but also to a configurable period afterwards, usually one orbit period past the target time, to protect for trajectory Pc problems after plan completion. Likewise, an optional maximum IMD constraint adds energy whenever two modules go beyond a prescribed upper range limit. When applied, this maximum IMD supports inter-module communication needs and can also be applied during or after the maneuver planning window. The scatter restrictions are implemented by adding energy for each module that encroaches into the 10 km scatter keep-out regions after the 5 minute time limit.

The selectivity and weighting of these constraints and objectives allow for management of competing operational needs while minimizing module delta-V and cluster Pc. For example, operational tradeoffs between reduced per-module delta-V and temporary loss of inter-module communication during scatter were explored by selectively disabling the maximum IMD constraint and intentionally allowing modules to drift farther apart during portions of each module's scatter maneuver plan.

3.4. SA Neighbor Selection and Annealing Schedule

The SA neighbor selection function is designed to have an equal chance to change any part of the SA state that can be searched, i.e. any variable element for any module. The range of variation for each state element is held fixed over the entire problem. However, two of the SA search states for each module, *Ti* and *Tf*, are not fully independent because *Tf* > *Ti*. Additionally, the operational capabilities of each module's attitude determination and control systems (ADCS) and propulsion systems drives a minimum wait period between burns, ΔT_{burn_wait}. This wait period also exists between the minimum *Ti* and *Tf* period and implies an additional restriction such that

$$T_f \geq T_i + \Delta T_{burn\_wait} \tag{2}$$

Rather than attempt to quantify a resulting system energy from an impossible search state, such as where *Ti* > *Tf*, the neighbor selection algorithm skips those possibilities and randomly re-selects a new element to change. This effectively alters the statistics of some state selection when *Ti* or *Tf* are close in value until the confining variable is adjusted, or relaxed. Enforcing this condition is not a significant hindrance to the overall SA search given that the allowable per-module *Ti* and *Tf* search ranges are large enough (half an orbit period or greater) to likely escape this condition.

For implementation and tuning simplicity, the SA search algorithm follows the classic fixed exponential annealing schedule based on the initial temperature, T_0, and a constant cooldown rate, ΔT, that is applied each time the temperature is adjusted,

$$T' = (\Delta T)T \tag{3}$$

where is $\Delta T < 1.0$. The SA algorithm transitions to a new temperature based on several standard criteria. The search is limited to a maximum number of iterations to conduct at each temperature and represents the "best effort" at each temperature. The number of times this "best effort" has been expended is also tracked and limited. The entire SA search is limited to a maximum number of iterations so that there is always a fixed upper length to the search. This also provides a separate qualification of the maximum effort before aborting the search. The number of accepted solutions, where *e'* < *e* or when *e'* > *e* was randomly selected, is tracked at each temperature level as well. The search advances with another step cooling after many solutions are accepted. Transitioning to the next temperature based on this success represents the "minimum search effort" at each temperature.

By contrast it should be noted that there are many possible variations on the solution acceptance criteria, neighbor selection function, and annealing schedule including those that dynamically accommodate the current energy and temperature [9]. The theoretical application of a well-tuned SA will find the global optimum given enough searching, however given that SA is allowed to revisit solutions, this search effort could likely be greater than an exhaustive brute-force search.

3.5. Multiple-Burn Solver

Each iteration of SA is supported by the multiple-burn solver (MBS) that generates a set of maneuvers to solve the boundary value problem specified by SA. MBS solves maneuvers for each module independently. SA provides MBS with the inertial position and velocity of a module at an initial time, the desired inertial position and velocity of the module at a target time, and a set of configurable burn parameters. MBS returns a set of burns in the inertial coordinate frame. The computed burns may be modeled and reported as impulsive or finite burns based on a configurable description of the module's capabilities. The solver also enforces a configurable minimum and maximum per-burn delta-V limit. The SA and MBS together accommodate the minimum wait period between burns, ΔT_{burn_wait}, by selecting the candidate burn times that drive the MBS selection of both impulsive and finite burns.

Since cluster flight maneuver planning often involves multiple orbit periods, multiple burn approaches significantly reduce delta-V use compared to two-burn targeters under many cluster flight circumstances. Computational complexity is a significant criterion, and for some methods became an important issue, especially for the use of shooting methods which tend to yield the best targeting accuracy. A compromise solution was driven by computational limitations and the wide scope of maneuver planning problems, which involve relative ranges of tens of meters to hundreds of kilometers. The original proof of concept for MBS solved a LP that was formulated using CW equations of motion based upon a reference orbit near the center of the cluster. This approach was adapted from [10]. LP was selected because it offers fast computation of optimal multiple burn solutions. The current MBS algorithm was created after performing a significant trade study. This study explored alternatives that improved accuracy and maintained similar delta-V performance under a wide variety of representative cluster flight conditions including large relative ranges from the cluster reference orbit.

The current MBS algorithm formulates the targeting problem in the local-vertical, curvilinear (LVC) coordinate frame of the target inertial orbit. The problem is formulated as the following convex problem:

$$\begin{aligned} &\textit{minimize} \left\|\delta v_1\right\|_1 + \ldots + \left\|\delta v_N\right\|_1 \\ &\textit{subject to :} \\ &\mathbf{A_v}(x_f, t_0, t_f) v_0 + \mathbf{A_r}(x_f, t_0, t_f) r_0 + \mathbf{A_v}(x_f, t_{v1}, t_f)\delta v_1 + \ldots + \mathbf{A_v}(x_f, t_{vN}, t_f)\delta v_N = \mathbf{0} \end{aligned} \tag{4}$$

The dynamics are formulated within the LVC frame relative to the final target state x_f such that the target state is at the origin with **0** representing the 6x1 zero vector. In addition, δv_n is the 3x1 nth impulsive burn in the LVC frame, t_{vn} is the time of the nth burn, N is the total number of burns. x_f is the 6x1 target orbit position and velocity in inertial coordinates and t_f is the target time. The initial conditions are specified within the LVC frame of x_f with r_0 representing the 3x1 initial position of the module at t_0 and v_0 representing the 3x1 initial velocity of the module at t_0. The transformation matrices [$\mathbf{A_r}(x_f,t_1,t_2)$ $\mathbf{A_v}(x_f,t_1,t_2)$] represent the 6x6 geometric derivation of a linear state transition matrix from time t_1 to time t_2 that includes eccentricity of the target inertial orbit and J2 gravitational effects [11,12]. $\mathbf{A_r}$ is the left side for transitioning the position state and $\mathbf{A_v}$ is the right side for transitioning the velocity state. Other orbital perturbations such as atmospheric drag, solar radiation pressure, and other gravity terms are not currently incorporated

within MBS. This convex problem can be formulated as a LP, which is solved using the simplex method [6, 7]. The result is a set of multiple impulsive maneuvers that are represented in the inertial frame. Maneuvers larger than a specified magnitude are recomputed as finite burns for higher-fidelity module execution. If there are overlaps of finite burns, then the candidate maneuver times are adjusted before the LP is reformulated and re-solved.

3.6. Inertial Propagator

The SA and MBS algorithms utilize an inertial orbit propagator based on the IAU-2000 CIO transformations [14]. It utilizes the JGM-3 gravity model up to degree and order 20 and an optional simple exponential atmospheric density model [15]. A fixed step Runge-Kutta integrator can be selected from 4^{th}, 5^{th}, 6^{th}, and 8^{th}, order [16, 17]. The propagator can propagate forwards and backwards in time, and it can propagate both finite burns and impulsive burns in maneuver plans.

3.7. Additional Operational Considerations

Applying the SA heuristic search is successful for searching out operational feasible solutions, but it has disadvantages as well as advantages. A well-tuned classic SA can be robust in finding the global optimum given enough iteration. However, the chief disadvantage of using SA is that generally it can take many iterations to converge to an equilibrium solution. Once a local region with a better solution is found, the SA does not necessarily directly progress towards the local minimum unless T is low. This complex behavior, while "obvious" to engineers in hindsight, is complicated by the fact that the problem spaces encountered by this application differ greatly in scale and may contain significant variation across the unknown search space for each different request. Given the scale of the search problem, only our most trivial problems could be searched well enough that statistically a global optimum could be found before the results are required by the MPS caller. The preliminary SA tunings we have used to date are more akin to a rapid "quenching" of the temperature that drives the search towards a local minimum after some exploration of the search space [9]. This is a sub-optimal use of classic SA, but this allows the same SA algorithm to produce feasible and operationally useful results under these significant time constraints.

Note that the single SA state after any given iteration does not necessarily have to be, and generally is not, the best solution ever encountered. A common strategy of retaining the best encountered solution allows MPS to abort the search early when the allowed time limit has elapsed. This also provides a form of system fall-back when an algorithm or software error occurs because then an existing earlier operationally feasible result can still be safely returned.

The single SA system-wide state allows for all modules to be planned simultaneously even with completely different maneuver types, for example planning ingress and station-keeping modules together. There are other ways to utilize these same algorithms. During CFA closed-loop re-planning efforts, a prior maneuver plan solution could be used to seed a new SA state and maneuver plan solution. Generally the new re-planning problem domain resembles the prior domain enough that the original solution is at or close to a new good solution. This allows SA to find a reasonably useful result more quickly than with a blind start. This is often the case with

station-keeping, ingress, and egress solutions. Scatter maneuver planning often involves a very large and rugged solution space where the benefits of starting with a prior solution could be even greater. [18] describes approaches to scatter maneuver planning.

Another method is an explicit use of MPS and SA to plan only a subset of modules in the cluster, or one module, at a time but also include the plans for prior module maneuver plans. Compared to a single maneuver planning effort that simultaneously accommodates all modules, this layered plan construction has two effects. The first effect is that the SA search space is limited and involves fewer, or only one, modules and therefore progresses more quickly. The second effect is that modules in the earlier plans ignored the effects of later modules and were given preferential treatment of their solutions. This can be operationally useful for planning nominal station-keeping for one subset of modules and then layering another plan for an ingressing module. The ingressing module has to avoid the station-keeping modules while the station-keeping modules do not have their plans (and delta-V use) perturbed by the presence of the ingressing module.

It is important to remember that operational concerns are not simply mapped into SA fitness function constraints and objectives. Many operational concerns are realized as direct search space limitations, others as neighbor selection limits, and some as restrictions in the MBS. SA state space limitation and discretization to reflect operational objectives greatly reduce the number of search space candidates. Overall method scalability is a function of the number of modules involved in planning, each module's search space discretization for the type of maneuver, and the computational cost of the constraint and objective evaluation including the MBS and propagator. Significant data caching is used to eliminate most re-computation at each iteration, but some inter-module relationships must still be re-calculated as a module's trajectory changes. With inter-module constraint evaluation disabled and all modules planned with an identical number of search space candidates, the algorithm scalability is expected to be linear, or $O(n)$, where n is the number of modules, to obtain equivalent results in the mean of many runs. Practically speaking, with all constraints enabled, the algorithm scalability is expected to be roughly quadratic $O(n^2)$, assuming similarly sized search spaces for each module and that the constraint evaluation takes much less time than the burn solver and propagator for each iteration as is currently the case.

4. Service Design and Implementation

4.1. Software Description

During development and eventual deployment, MPS and other CFA services were always expected to be hosted and tested in different environments. A reusable core software private implementation is built with C++ and defines the service interface functions and data types. This core is isolated behind a service adaptor and several supporting interfaces. The service adaptor and host application meet the deployment environment needs without requiring changes to the private implementation behind the adaptor. The MPS service state machine logic, error handling, commanding, telemetry, and message processing are handled in hand-coded C++. The C++ code calls the above algorithms and their data handling logic. These algorithms are written in MATLAB functions which are converted into C source code generated using MATLAB

Coder. Key interfaces for logging, file input/output, and system time are used to further insulate CFA applications across the various deployment environments.

MPS operates in a message-based environment. Most application inputs and outputs are structured as messages through the MPS adaptor. Some external interactions, such as application logging, file input/output, and access to the system time are written to interfaces. Those can be implemented as messages or direct system calls depending on the actual deployment environment without changing the reusable code. The non-reusable portions of MPS have little functionality other than basic message publish/subscribe and message data mapping, the application execution loop, and supporting the logging, file input/output, and system time interfaces. In total, this is a small fraction of software compared to the reusable portion of MPS. To accommodate message passing application design, MPS supports lightweight message reception methods and performs most of the heavy lifting in a separate main processing function call. Figure 5 shows an example MPS deployment instance as a composite structure diagram. The left-side pins show the incoming maneuver plan requests and updates to the MPS configuration as well as the outgoing maneuver plan results and MPS status (which include service telemetry). Other MPS command messages which do not involve these algorithms are not shown. The gray-shaded elements are specific to a deployment while the other elements are reusable code. MPSPrivate, libmps_alg, and other supporting classes (not shown) are the reusable private implementation. Note that libmps_alg houses MATLAB-based algorithms in C source code as a separate library. The MPSAdaptorIF, MPSAdaptor, MPSLogger, FileIO, and CFATime are abstract classes that form the reusability boundary for MPS. Concrete implementations of those may be unique to each deployment.

MPS is built to process one maneuver planning request message at a time. The MPS state machine configures the service at startup then waits for a request. After a request is received, the state machine invokes various algorithms to validate the request, translate it from CFA maneuver planning terminology into the SA search domain, execute the SA search, then translate the SA result into an outgoing maneuver planning result message. It handles errors encountered at each step in this process and periodically sends telemetry to the ground for MPS monitoring. Once a request has been processed MPS is ready for another request. MPS is designed to maintain responsiveness so that it halts current processing and returns the best result found if it receives a request during processing.

4.2. Software Configuration and Algorithm Tuning

MPS software configuration is split into service configuration and algorithm tuning parameters. The service configuration covers identification of the MPS instance, controls over logging (the amount and type of data to log), and controls over telemetry (enable/disable and publishing period by telemetry type). The service configuration also includes platform tuning which is used to control how much processing MPS attempts on each processing cycle on a given deployment platform. The algorithm tuning covers the propagator fidelity, how often MPS performs certain constraint checks, and sets of SA configuration parameters for the different maneuver planning types. This MPS software configuration can be inspected or updated at any time by the ground operator via messages. Configuration updates that occur while MPS is processing a request are held until the processing completes and before the next request processing commences.

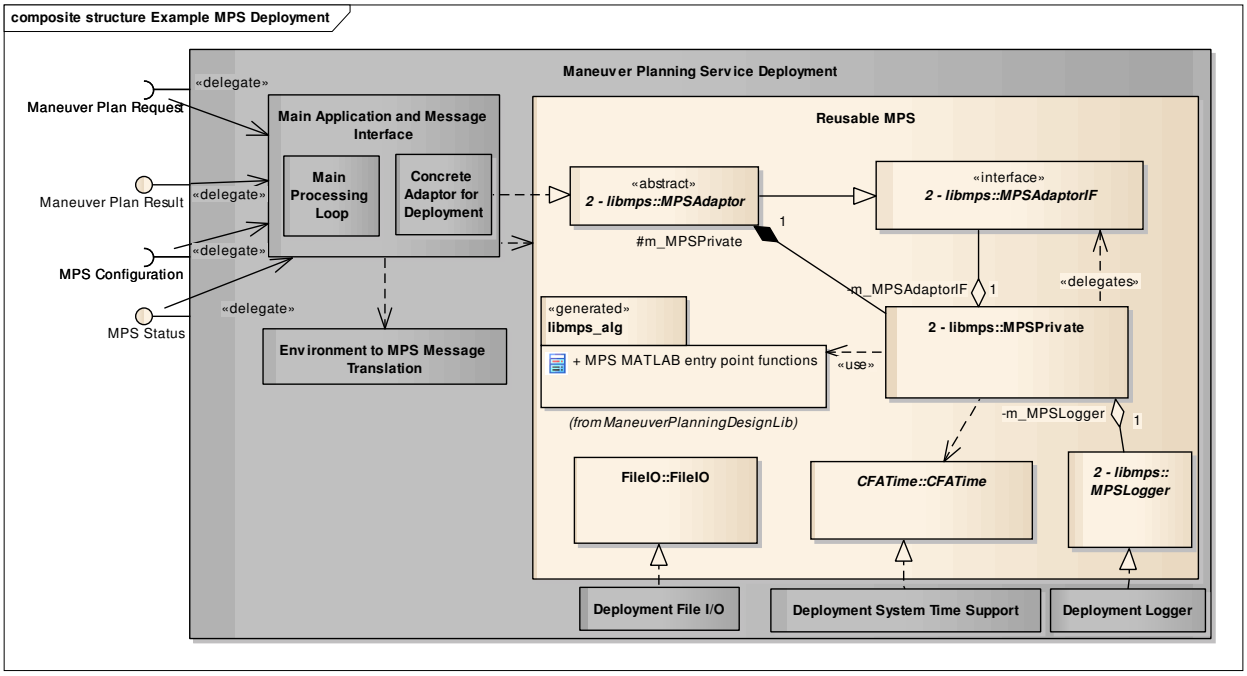

Figure 5. Example MPS Deployment (Composite Structure Diagram)

4.3. Software Implementation and Use

Initial development began on the x86_64 Linux platform for expediency and well before any target platforms were identified. ANSI C++ (C++03) and ANSI C (C99) are used with MATLAB Coder and a custom Pilot Support Project from The MathWorks, Inc. to help manage the C header structure member types in MATLAB Coder. No dynamic memory allocation, C++ exceptions, or C++ templates are used. Array lengths are fixed and most operations outside of file input/output are endian-agnostic. The current algorithms require IEEE-754 double precision floating point support for accuracy and hardware support is recommended for computational performance. The codebase has been converted to target 32-bit platforms (i686 and ARMv6 CPUs have been tested to date) with minimal source code changes.

The reusable portion of the MPS service has been successfully deployed into multiple environments throughout development. MPS was initially deployed as an application in a component model environment on Linux from Vanderbilt University [19] and paired with Emergent's orbital environment simulation [20]. MPS ran at wall-clock speed and demonstrated basic message passing and successful algorithm behavior. Later MPS was hosted in the DIECAST Software-In-the-Loop (SIL) simulation which ran faster than wall-clock speed for long-term CFA scenario testing and demonstrations [20]. Other test harnesses have been built for MPS-specific testing as standalone Linux applications, with the Google C++ Testing Framework, and as wrapped MATLAB MEX functions. MPS has also been deployed as an application in NASA's Core Flight Executive (cFE) and Core Flight Software System (CFS) architecture running on Linux.

5. Results

We present example results for three significant types of maneuver planning that represent the SA algorithms and MPS software capabilities: ingress, cluster reconfigure, and defensive scatter with subsequent regather. The solution space of each given example is progressively more challenging. By presenting these cases, we intend to demonstrate the robustness and flexibility of these algorithms and software to challenging problems. Additional aspects of cluster flight station-keeping strategies are described in [21].

5.1. Cluster Ingress

We first present an operationally representative example for maneuver planning a single module ingressing into an existing two module cluster in a circular 600 km altitude Earth orbit. The two cluster modules (modules 1 and 2) already have their station-keeping maneuver plans calculated so the SA only has to search for solutions involving the third single module. Figure 6 shows the initial, transfer, and final trajectories. The green circles show each module's starting position and the red circles show each module's final position. Modules 1 (green trajectory) and 2 (blue trajectory) are in a cluster configuration with similar ROEs differing only in their β angles. Module 3 (red) starts with an ROE designed for passive safety relative to the existing cluster (larger a_e and z_{max}) and positioned in an initial relative parking orbit ahead of the cluster. Module 3 ingresses from its parking orbit into a similar cluster ROE as modules 1 and 2 with a different β angle.

Even though only one module is being maneuvered, all relevant constraints and objectives are evaluated including minimum and maximum IMD between all modules. The search space for the ingressing module was discretized into 10 elements for *Ti* and 38 elements for *Tf*, over an allowable maneuver planning window of two orbit periods. The desired target ROEs were fixed in this search so only these two search state elements varied. Out of the 380 possible search space combinations, the SA search visited 60 unique states after a 100 iteration limit. This 100 iteration search took 1.6 seconds on a 2.5GHz Intel Core i7 laptop running a 32-bit MPS service under a 64-bit x86 Linux Virtual Machine instance.

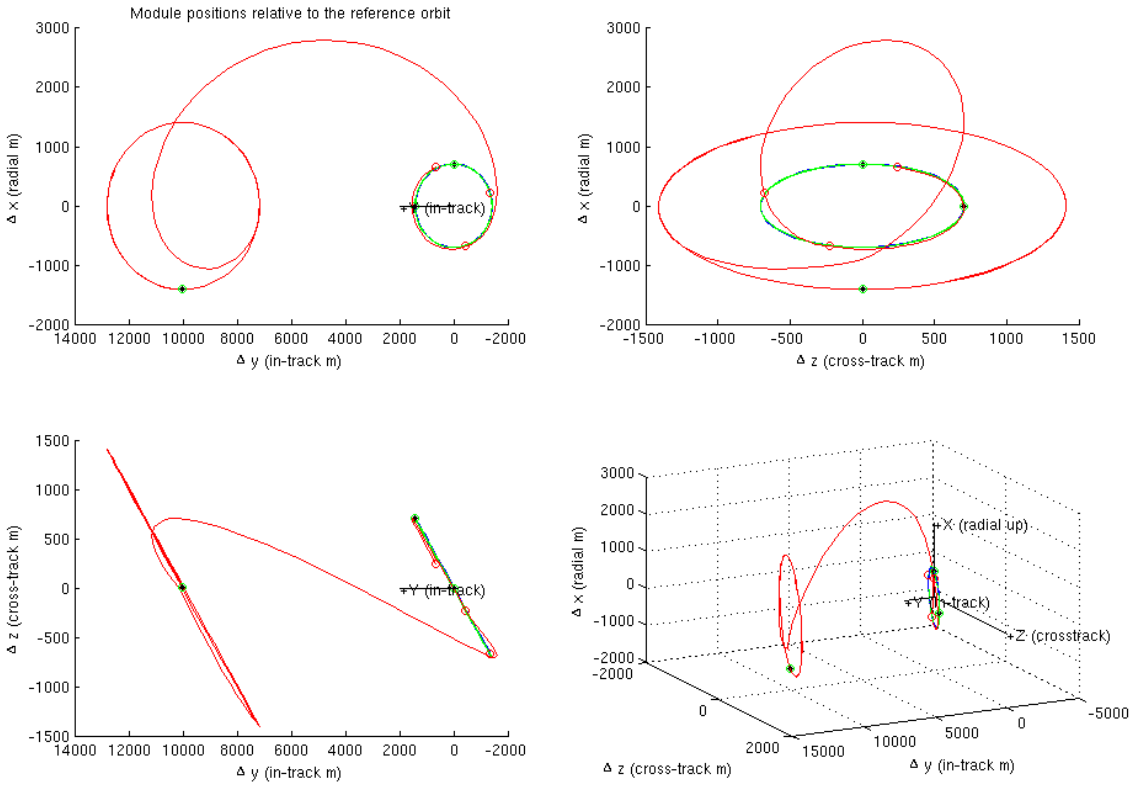

Figure 6. Single-Module Ingress: Initial, Ingress, and Final Orbits.

As can be seen in Fig. 7(a) the SA energy over the explored search space was relatively smooth with no infeasible states. Infeasible states are where $e > 1.0$. The variation in total energy is entirely due to the different delta-V maneuver plans among the various solutions. This was expected due to the operational setup. The ingressing module was pre-positioned with significant range to keep Pc low and in phase in with the planned relative ingress final ROE location. The SA search was tuned to rapidly lower the temperature and converge. The best solution was found on iteration 68 and involved six maneuvers with a total of 1.928 m/s delta-V. Figure 7(b) shows that several solutions were found in early iterations which had similar delta-V use. Note that many solutions were accepted throughout and SA candidates were only rejected at low temperatures. This search space was relatively smooth and contained a single search space minimum. Different starting temperatures and annealing schedules yielded the same solution for this problem. This example demonstrates a somewhat aggressive "quenching" annealing schedule but one that has proven to be robust with few modules and non-scatter maneuvers. The initial SA temperature, T_0, was chosen for expected robustness for this type of problem. Tuning for a more rapid annealing schedule and from a lower T_0 is possible for this problem but is not robust to more rugged search spaces. For this problem, the best solution happens to be found at the edge of the allowable *Tf* search space. It is likely that a better (lower delta-V) solution could be found by relaxing the given operational maneuver planning window limitation and extending the *Tf* search space to allow the module to take longer than two orbits to transfer to its target orbit. This case is very representative of maneuver planning that is operationally benign, without constraint violations, and represents most station-keeping and ingress type of problems.

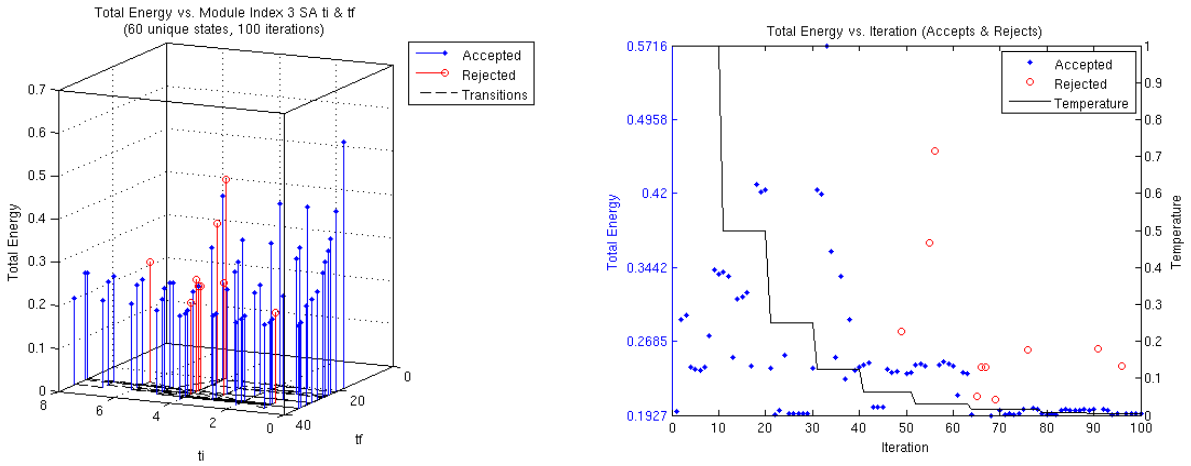

Figure 7 a & b. Single-Module Ingress Example: Search Coverage (a, left), SA Energy and Temperature (b, right)

5.2. Cluster Reconfiguration

We next present a more challenging example of simultaneous maneuver planning involving three modules. In this scenario, all three modules are in relative orbits that are quite different from the desired configuration. To make the problem even more interesting, the modules are phasing away from the cluster reference orbit and desired reconfiguration origin. Figure 8 shows the future trajectories of the modules if no maneuvers would be performed. Module 1, blue, starts near the origin with an ROE with little cross-track component. Module 2, green, is in the medium-sized ROE. Module 3, red, has a very large ROE and starts at +10km in-track. The cluster reference is again a circular 600 km altitude Earth orbit. This maneuver problem will require more aggressive maneuvering to reconfigure each module orbit in four orbit periods. All non-scatter constraints and objectives are evaluated including minimum and maximum IMD between all modules. Please note that arresting relative motion and reconfiguration of a cluster at the same time is operationally unlikely, but it is an excellent demonstration of the robustness and capabilities of this maneuver planning approach.

The search space for each of the reconfiguring modules was discretized into 10 elements for *Ti* and 120 elements for *Tf*, over an allowable maneuver planning window of four orbit periods. The desired ROEs for each module were fixed in this search so only two search state elements varied. This gave 1,240 unique search space combinations for each module and over 1.9 billion unique search space combinations. The SA search visited between 114-135 unique states for each module over a 600 iteration limit. This 600 iteration search took 8.7 seconds on a 2.5GHz Intel Core i7 laptop running a 64-bit x86 Linux Virtual Machine instance.

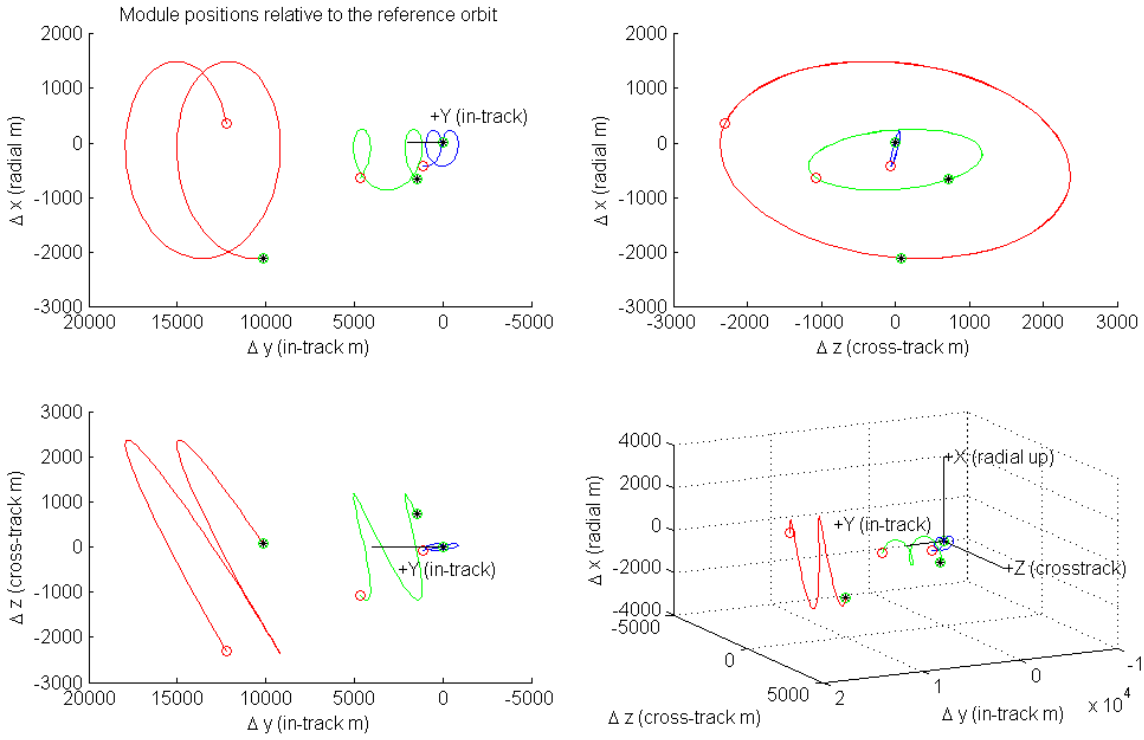

Figure 8. Cluster Reconfiguration Initial Projected Orbits.

Figure 9 begins at the same time as Fig. 8 but shows the resulting transfer orbits and final orbits. Let's first examine module 3, red, in the upper-left radial-in track plot. The MBS and SA found a three-burn solution for module 3 with its first burn after waiting over half an orbit from the problem start. This burn began a phasing trajectory back towards its designated ROE. Two more burns placed the module on the target ROE. Note the reconfigured ROE, closed ellipse near the origin, for module 3 is a completely different size, shape, and orientation.

We can observe module 2, green, and module 1, blue, trajectories in the zoomed-in views of Fig. 10. Module 2 conducts three maneuvers. It waits to perform its first burn until around +4500m in-track and establishes a closing transfer orbit. A second burn, around +2000m in-track adjusts its cross-track phasing and a final burn places it on its destination ROE. The in-track-radial and in-track-cross track plots show that module 2 and 3 trajectories never place the modules close to each other at the same time. Module 1, which starts near the origin with a non-zero velocity, reconfigures to the cluster reference orbit. It immediately arrests its opening rate then phases to arrive at the origin before executing final maneuvers to arrest its remaining relative motion.

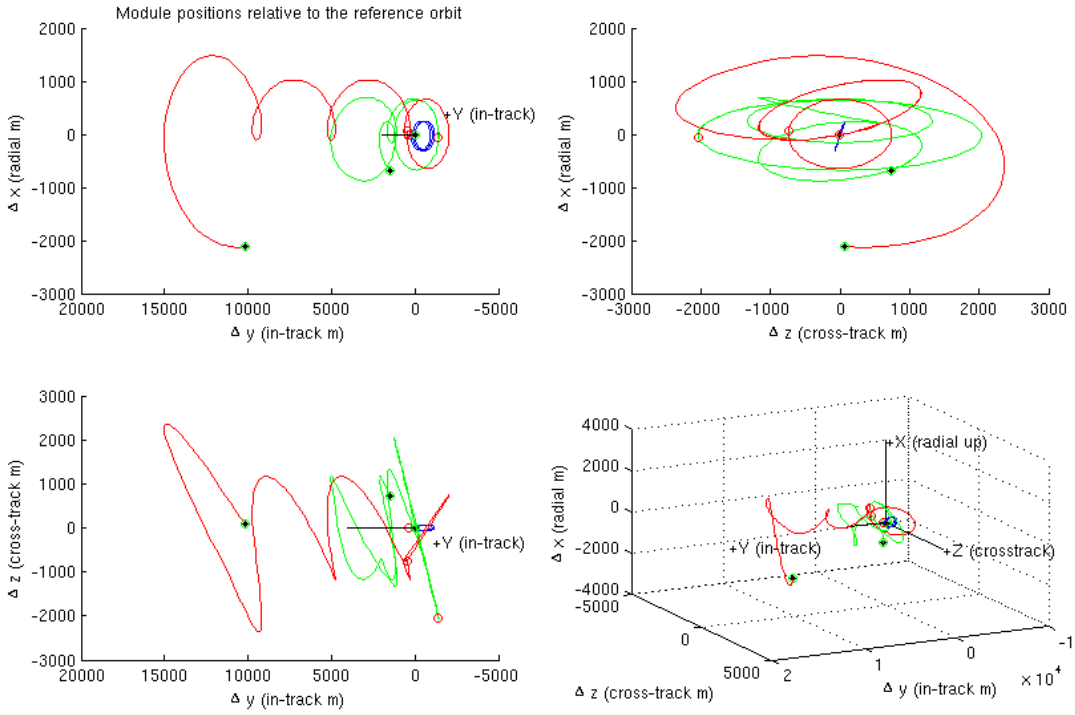

Figure 9. Cluster Reconfiguration Transfer Orbits.

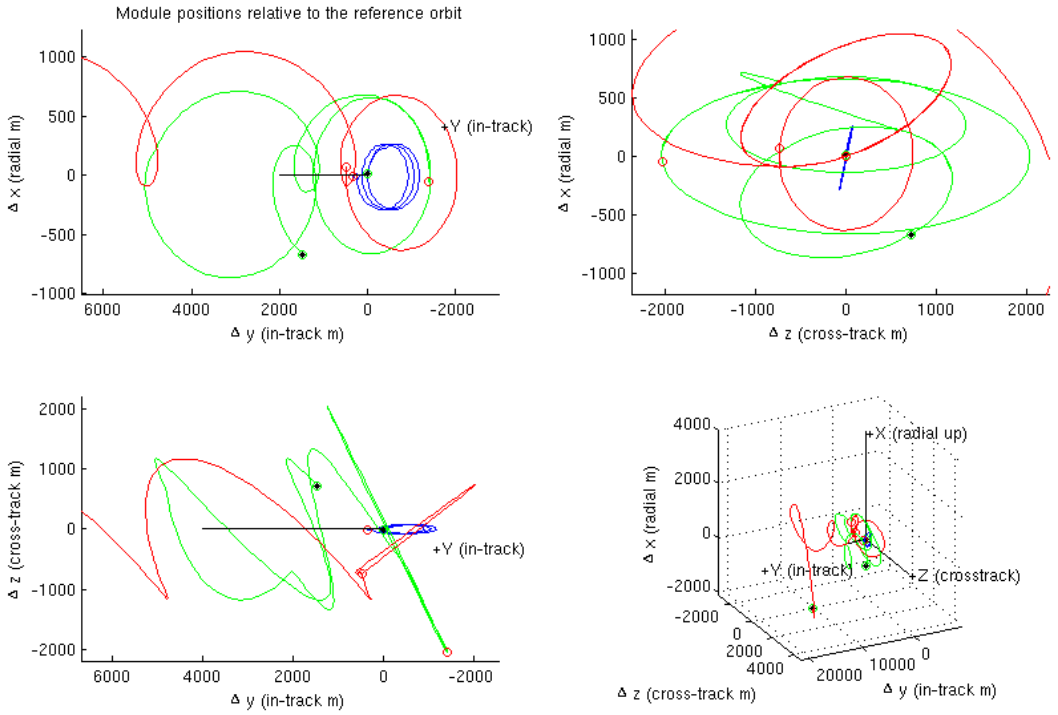

Figure 10. Cluster Reconfiguration Transfer Orbits (zoom).

Figure 11 shows the final configuration of the cluster. Module 1 is located at the origin. Again, while these trajectories are extreme from an operational point of view, it is important to note that this is a sizeable volume of space and the maneuver planning algorithms check the IMD to keep Pc acceptably low. Application of these IMD constraints does affect the problem search space.

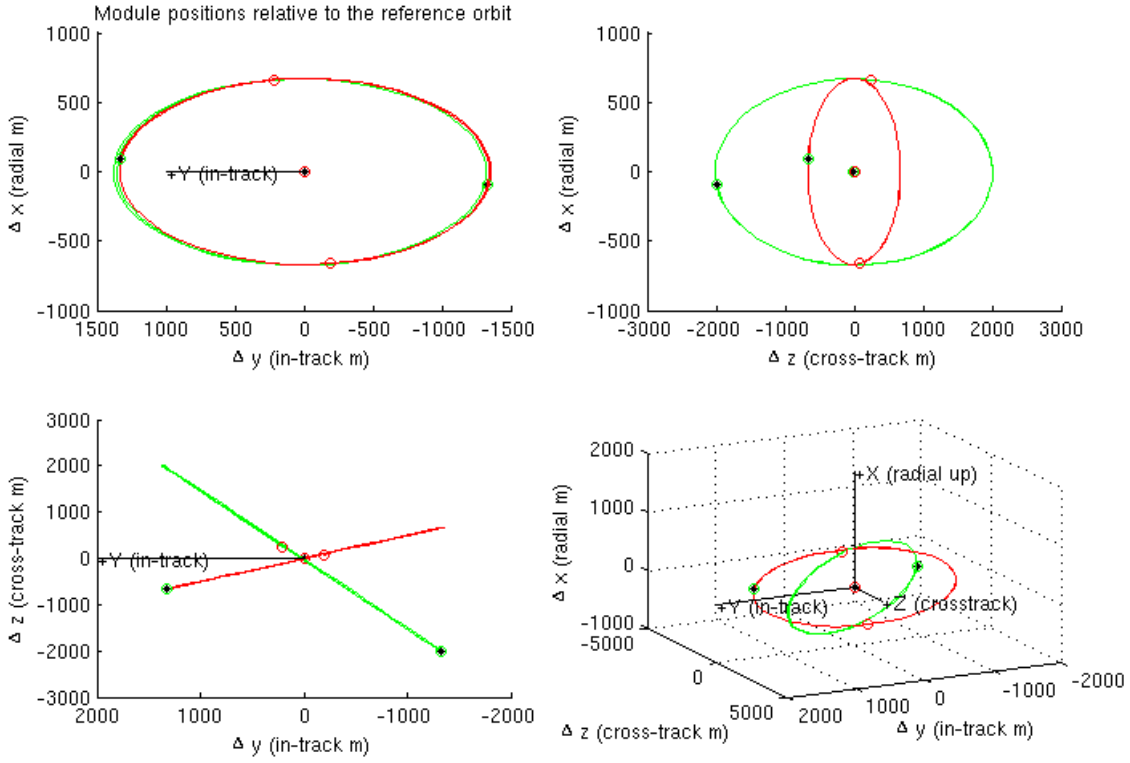

Figure 11. Cluster Reconfiguration Final Orbits.

Figure 12 shows the unique *Ti* vs. *Tf* state combinations for two of the three modules. This search problem space had much more variation than the prior ingress example. A total of 40 infeasible solutions were encountered during the search with SA energy levels up to 100.3 as compared to $e < 1.0$ for feasible solutions. Most of these are due to IMD violations. Note that some per-module state combinations have multiple energy levels and multiple acceptance/rejection by the SA search. This corresponds to state variations in another module's state elements while this module's state was the same value.

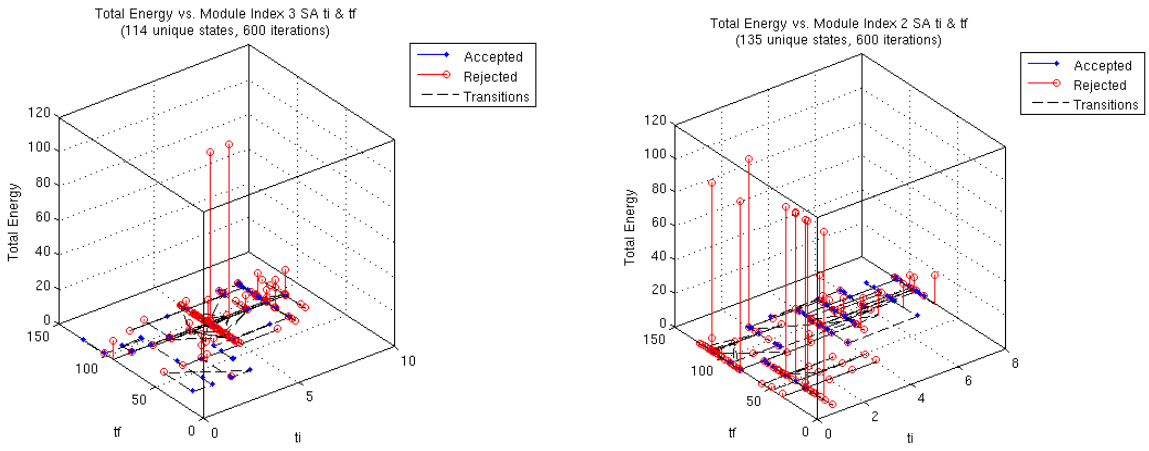

Figure 12. Cluster Reconfiguration Example - Search Coverage for Two Modules (of Three)

Figure 13 presents the same data as Fig. 12, at a reduced scale, showing only the feasible solution energy plotted against each module's state elements. Even when neglecting the significant infeasible solutions, compared to the ingress Fig. 7(a), there is more variation in the feasibility across the search space.

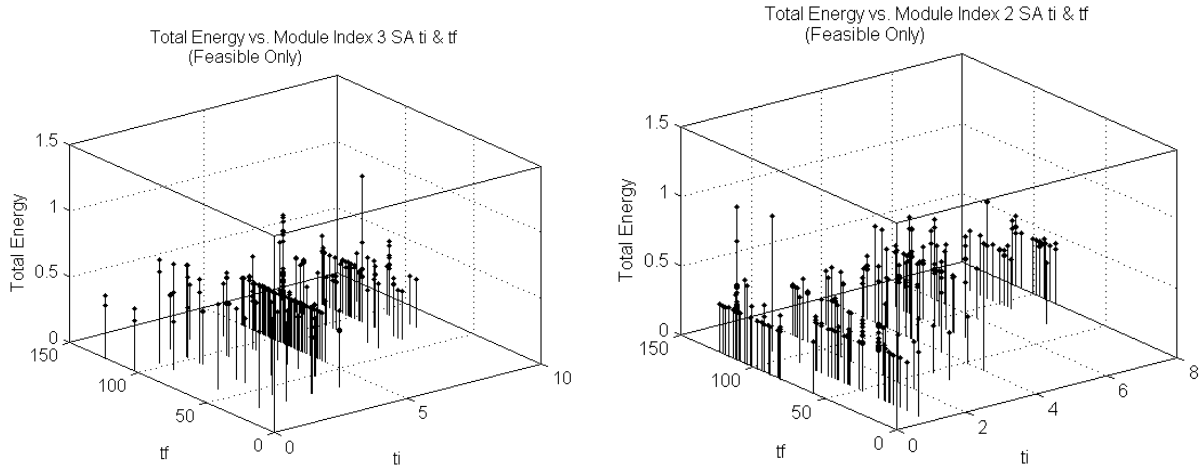

Figure 13. Cluster Reconfiguration Example - Feasible Search Coverage for Two Modules (of Three)

Figure 14(a) shows the SA search energy for each iteration with the infeasible solutions at values having the most energy. Note the log scale. Figure 14(b) shows the same but zoomed in with a linear scale. The SA search was tuned to lower the temperature and converge slightly less rapidly than the prior ingress example but allowed to run six times longer since there was a larger search space. The best solution was found on iteration 595 (out of 600) with an energy value of 0.337. However, despite the more rugged search space, again many operationally useful solutions were found in early iterations.

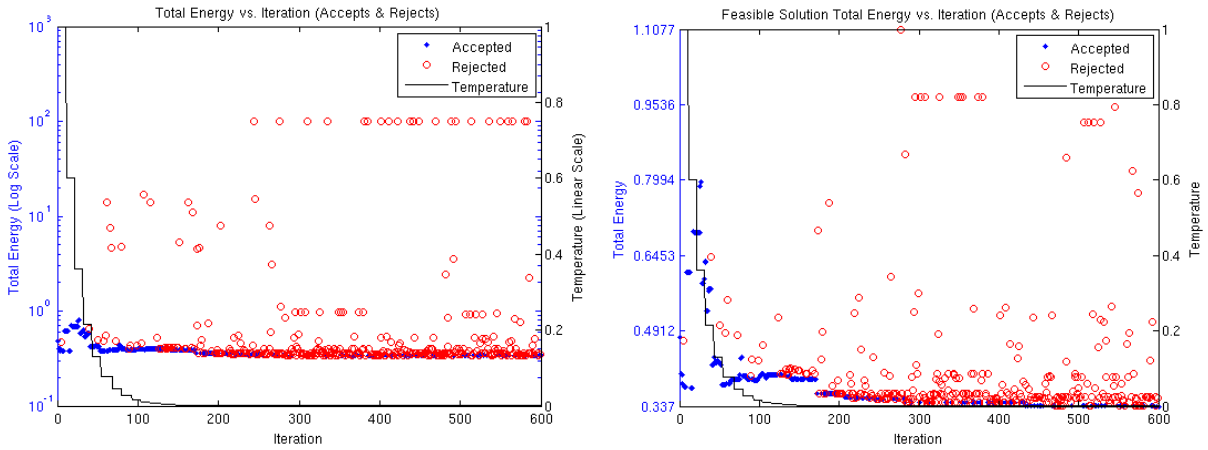

Figure 14 a & b. Cluster Reconfiguration Example: SA Energy and Temperature, Full semi-log plot (left, a), zoomed-in linear plot (right, b)

This cluster reconfiguration example used the same algorithms and software as the prior ingress example with only a minor change to the configurable annealing schedule. These same algorithms are quite capable of developing robust maneuver solutions to difficult maneuver planning problems while maintaining operational safety.

5.3. Scatter and Regather

DARPA's System F6 initially had a specific requirement to defensively scatter the modules of a cluster so that they are not within 10 km of any of their original, un-scattered, trajectories in 5 minutes in order to avoid threats. However, once the modules have used significant delta-V to scatter, they must eventually be regathered back into an operational cluster. In fact, if the module scatter maneuvers are not properly coordinated, then the time and delta-V required to regather the cluster can become prohibitive. As discussed more thoroughly in [18], the CFA conducts scatter maneuvers and subsequent regather maneuvers in stages with multiple maneuver plans.

Typically the modules are scattered to a set of "hold orbits" that satisfy the 10 km keep-out constraint from the original module trajectories. Hold orbits allow the modules to scatter into a larger volume of space to minimize Pc without requiring high-fidelity or risky maneuvers for directly inserting modules back into the original cluster configuration [18]. Hold orbits allow the CFA time to correct any module trajectory errors from scatter maneuvers.

The desired regather location for the new cluster and the regathered cluster geometry determines the ROE target orbit search ranges of the possible hold orbits. For example, the modules must arrive from the initial scatter into their hold orbits and still be within communications range of each other. The hold orbit ROE search space is range-limited and discretized to minimize the search space while setting up the modules for efficient subsequent regather operations. Additionally, mismatched ROE β angles generally incur a significant delta-V penalty compared to differences in other elements. Therefore, the desired regather ROE β angle is used as a scatter hold orbit target for each module. Finally, the hold orbits are designed such that the modules would be passively safe in the radial crosstrack plane with respect to the regathered cluster

configuration. Generally MPS searches *Tf*, the ROE a_e, z_{max}, and y_d, as well as the scatter target point (az, el, and range) for each scattering module.

The CFA directs all modules to burn as soon as possible after the scatter command is received. This is possible because MPS is used to pre-compute the scatter maneuver plans for potential use. This allows each module's delta-V use to be tailored to its trajectory and scatter constraints. As the scatter progresses, CFA interacts with MPS to correct trajectory errors, as is done with station-keeping, in order to meet the hold orbits.

Since scatter and regather are not current System F6 objectives, a prior version of the MPS software was used to generate these results, where the LP MBS uses simplified CW equations of motion and impulsive burns with an unperturbed central-body gravity model. It is important to note that other than this, all of the current MPS algorithms and much of the software are the same or trace direct lineage. The current MPS software operates on a data model designed to accommodate scatter maneuver planning, and thus we feel this example is representative of this approach.

Figure 15 shows the trajectories for a 20-module initial cluster configuration scattering to a set of hold orbits centered around a -300 km position on the in-track axis. The hold orbits are within +/- 40 km of this position, and the modules are constrained to always be inside a 100 km maximum range between any two modules to support a notional inter-module communications range limit. For this case, the maneuver planner was configured to allow maneuvers to occur up to 2.5 orbits after the initial scatter. Green circles around the origin are the original cluster module positions at t=0. Red circles are the module positions at t=2.5 orbits.

For this case and maneuver planning configuration, the modules scattered to their hold orbits between 1.96 and 2.38 orbit periods after the initial scatter. The mean delta-V for all 20 modules was 85.6 m/s with standard deviation of 9.7 m/s. The mean scatter delta-V and deviation are generally higher for clusters having more modules and when maximum inter-module constraints are levied on the transfer trajectories. With both conditions, the SA search has a larger and more difficult solution space due to the combinatorial effects on constraint evaluation of the relative module trajectories.

After modules have reached their hold orbits, CFA uses MPS to station-keep while planning cluster regather using a sequence of ingress maneuvers in one or more maneuver plans. Each of these regather maneuver plans can apply to one or more modules and is conducted to re-build the operational configuration of the cluster. Typically *Ti* and *Tf* are range-limited so that specific portions of the cluster are regathered in stages. This helps to ensure passive safety with the modules that are in hold orbits waiting to ingress back into the regathered configuration.

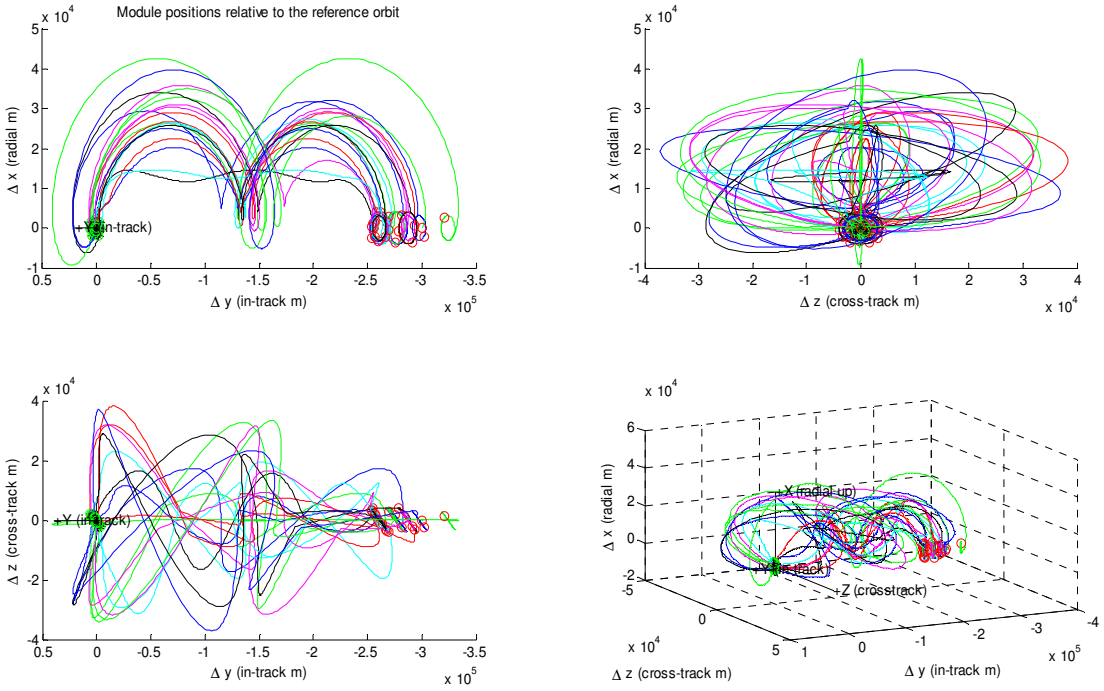

Figure 15. 20-Module Scatter to -300km In-track Hold Orbits

Figures 16 and Figure 17 depict the regather trajectories of these scattered modules from their hold orbits with respect to the desired regather location at the origin. We used MPS to regather the modules in two overlapping stages, each utilizing a single maneuver plan covering a total of five orbit periods. The two stages, both three orbit periods long, overlap for one orbit of the five-orbit regather period. This demonstrates how the second stage maneuver plan can be built upon the first stage plan. The final cluster configuration of 20 modules is designed with either a smaller or larger a_e-z_{max} ROE combination. Therefore the inner modules are regathered over the first three orbits as shown in Fig. 16. The upper-right radial/crosstrack view of Fig. 16 shows the initial inner module cluster orbits forming with a radius of just over 1 km. Note in this particular case, all but one module scattered to hold orbit locations ahead of the designated central regather location, but this is not always the case for every scatter problem.

Figure 17 shows the final two orbits of the regather where the remaining modules transfer into their final cluster configuration. All of the modules are either in their final locations or are transferring to their final locations. The upper-right radial/crosstrack view of Fig. 17 shows the module cluster orbits forming with radii between 1-2 km.

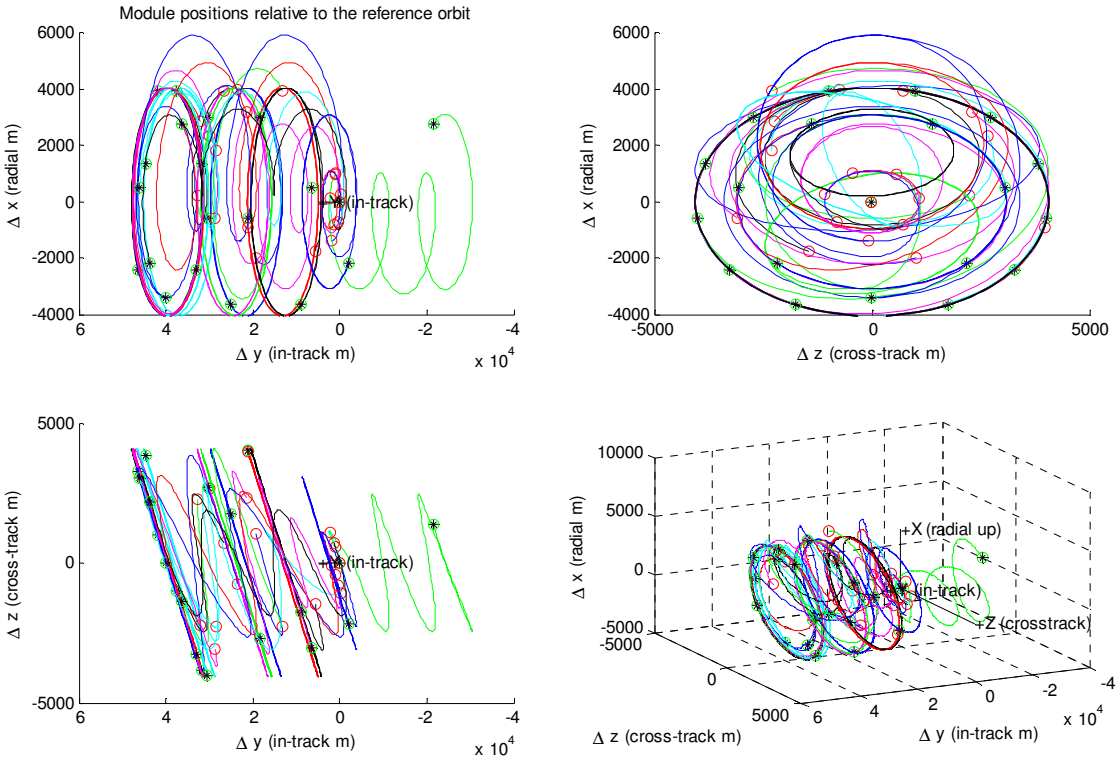

Figure 16. Regathering the 20-Module Cluster (Stage 1)

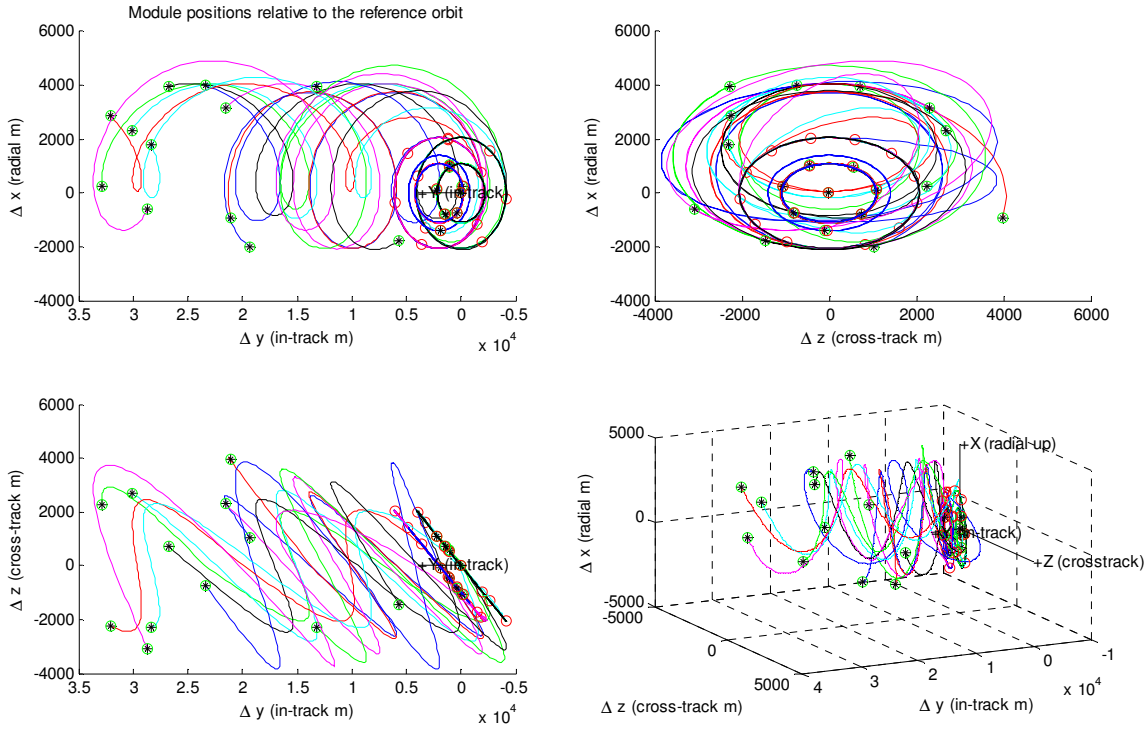

Figure 17. Regathering the 20-Module Cluster (Stage 2)

Figure 18 shows the final regathered cluster configuration. The mean delta-V for all 20 modules for these regather maneuvers is 3.1 m/s with a standard deviation of 0.9 m/s. Most modules regathered using multi-rev transfers between two and three orbit periods long.

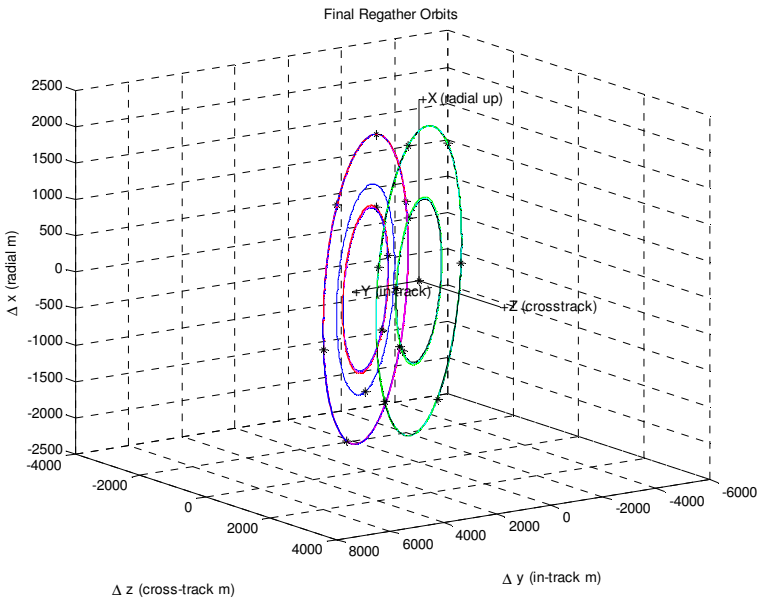

Figure 18. Regathering the 20-Module Cluster (Regathered)

6. Applicability and Possible Enhancements

This Maneuver Planning Service and algorithms have direct applicability to a variety of on-orbit maneuver planning problems involving one or multiple vehicles. This message-based service could be deployed on-orbit as well as on the ground and multiple instances could be used to provide fail-over redundancy or parallel computing. The current MPS could be used in concert with other trajectory planning approaches either as an initial solution generator or to refine a pre-computed solution using its ability to seed from a prior solution. Adding additional objectives or operational constraints is straightforward and would allow its use in new domains such as rendezvous and proximity operations or in more tightly-controlled formation flying. Also, current capabilities could be leveraged for different effects. For example, the scatter target point could apply additional control over trajectories for problems other than avoiding a scatter keep-out requirement. Multiple uses for control over the minimum and maximum inter-module distance constraint evaluation have already been identified.

The loose coupling of the internal algorithms allows for straightforward propagator or MBS upgrades with few impacts to other algorithms. Examples could be addition of other perturbation models such as higher-order gravity terms, atmospheric drag, solar radiation pressure or 3rd body gravity as well as for use around different central bodies. A direct upgrade from the current fixed-step integrator to a variable step or multiple-step integrator would likely increase the overall algorithm computational performance and propagator accuracy. The current MBS is a compromise to address the computational performance constraints as well as the needs of scatter and station-keeping, two very different problem domains. A likely upgrade is to create

several MBS algorithms, each of which is optimized for a particular problem domain and dynamically select them during the search.

The current software and earlier algorithm prototypes demonstrate the flexibility of this SA approach when applied to many cluster flight maneuver problems. Additional software flexibility for the maneuver problem setup and tuning will continue to increase runtime utility. The combination of different sets of SA problem setup and tuning with different fidelity multiple-burn solvers might allow a two-phased, or multi-phased, maneuver planning approach. Given the ability to create and ingest maneuver plans, MPS could be used in concert with other algorithms and systems outside of the CFA.

The SA algorithm has been studied for several years, with many refinements and extensions to this basic algorithm having been identified in a number of areas. Some of the more interesting SA upgrades are to use more "intelligent" variations of simulated annealing where the annealing schedule and neighbor selection functions are dynamic and might take into account the SA energy, temperature, number of iterations, etc. [9, 22]. Many of these improvements should aid search robustness and convergence. Another exciting possibility is to incorporate parallel or distributed simulated annealing [23, 24]. Approaches like these could spread the search problem across multiple processors to explore the search space more thoroughly. This is quite attractive for cluster flight where, as modules are added, multiple on-orbit MPS instances could take advantage of the growth in available processing resources to tackle the subsequently larger and more rugged problem domains.

As with all computationally intensive algorithms the choice of deployment platform, algorithms, and software optimization are key performance variables outside of the problem domain itself. Throughout development algorithm fidelity, memory footprint, and execution performance have been balanced with this in mind.

7. Conclusion

This paper has presented a flexible approach that satisfies all CFA cluster flight maneuver planning needs with a minimum number of different algorithms requiring implementation and validation. This paper has presented selected supporting results from these algorithms that demonstrate example cluster ingress, cluster reconfigure, and scatter with subsequent regather. The SA heuristic search, combined with the maneuver planning parameterization presented here, provides a robust approach. The LP-based multiple-burn solver provides a good compromise on fast execution and maneuver plan accuracy. These algorithms are implemented in a message-based software service that supports cluster flight station-keeping, ingress and egress of modules, reconfiguration of the cluster, defensive scatter and subsequent regather of the cluster.

With its reusable core behind designated interfaces, this service has been successfully rapidly prototyped, matured, and demonstrated in several orbital software-in-the-loop environments as well as in NASA's cFE and CFS architecture on Linux. This service and these algorithms have many avenues for direct expansion and upgrades and have applications beyond System F6 cluster flight.

8. Acknowledgements

The authors wish to thank MathWorks, Inc. for their support in utilizing MATLAB and MATLAB Coder on this project to generate target C source code from MATLAB functions. This support allowed algorithms to rapidly evolve from prototypes to compiled source code integrated in the C++ service applications and saved significant time and effort. The authors also acknowledge the contributions of Andrew T. Takano and Ricardo Restrepo.

9. References

[1] United States Government, Defense Advanced Research Projects Agency, Tactical Technology Office "System F6 Program." [Online] http://www.darpa.mil/Our_Work/TTO/Programs/System_F6.aspx [cited 25 Feb 2014].

[2] United States Government, Defense Advanced Research Projects Agency "System F6." Solicitation Number: DARPA-BAA-11-01, Oct 20, 2010 and amended.

[3] Hur-Diaz, S., and O'Connor, B. "Cluster Flight Application On System F6." *Proceedings 24th International Symposium on Space Flight Dynamics - 24th ISSFD*. Laurel, MD, May 2014.

[4] Cerny, V. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm." *Journal of Optimization Theory and Applications*, Vol. 45, No 1, pp.41-51, 1985.

[5] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by Simulated Annealing." *Science*, New Series, Vol 220, Number 4598, 1983, pp. 671-680.

[6] Luenberger, D., and Ye, Y. Linear and Nonlinear Programming. Springer, 2nd ed., 1984.

[7] Nelder, J. A., and Mead, R. "A Simplex Method for Function Minimization." *Computer Journal*, Vol. 7, No. 4, 1965, pp. 308-313.

[8] Lovell, T. A., and Tragesser, S. G. "Guidance for Relative Motion of Low Earth Orbit Spacecraft Based on Relative Orbit Elements." *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, RI, August 2004.

[9] Ingber, L. "Simulated annealing: Practice versus theory." *Mathematical and Computer Modelling*, Vol. 18 No. 11, 1993, pp.29-57.

[10] Tillerson, M., Inalhan, G., and How, J. "Co-ordination and Control of Distributed Spacecraft Systems using Convex Optimization Techniques." *International Journal of Robust and Nonlinear Control*, Vol. 12, 2002, pp. 207 242.

[11] Gim, D. W., and Alfriend, K. T. "State Transition Matrix of Relative Motion for the Perturbed Noncircular Reference Orbit." *Journal of Guidance, Control, and Dynamics*, Vol. 26, No. 6, 2003, pp. 956–971.

[12] Alfriend, K. T., Vadali, S. R., Gurfil, P., How, J., and Breger, L. Spacecraft Formation Flying. Elsevier Astrodynamics Series, 2009.

[14] Vallado, D. A., and McClain, W. D. Fundamentals of Astrodynamics and Applications. 3rd ed., Microcosm Press, 2007.

[15] "Global Gravity Field Models." International Centre for Global Earth Models (ICGEM) [Online], URL: http://icgem.gfz-potsdam.de/ICGEM/modelstab.html [cited 31 Jan 2013].

[16] Chapra, S. C., and Canale, R. P. Numerical Methods for Engineers. 5th ed., McGraw-Hill, New York, NY, 2006.

[17] Montenbruck, O., and Gill, E. Satellite Orbits: Models, Methods and Applications." 1st ed., Springer, The Netherlands, 2005.

[18] Duffy, B., Brown, A. G., Ruschmann, M. C., Ward, L. "Scatter Strategies for Cluster Flight." *Proceedings 24th International Symposium on Space Flight Dynamics - 24th ISSFD*. Laurel, MD, May 2014.

[19] Dubey, A., Karsai G., and Mahadevan, N, (2011) "A component model for hard real-time systems: CCM with ARINC-653." *Journal of Software: Practice and Experience*. 41, 12 (November 2011), 1517-1550. DOI=10.1002/spe.1083 Online] http://dx.doi.org/10.1002/spe.1083 [cited 26 Feb 2014].

[20] Stewart, S. M., Ward, L., and Strand, S. "Distributed GN&C Flight Software Simulation for Spacecraft Cluster Flight." *AAS Guidance Navigation and Control Conference*, Feb. 2014.

[21] Ruschmann, M. C., Duffy, B., de la Torre, R., and Hur-Diaz, S. "Efficient Station-Keeping For Cluster Flight." *Proceedings 24th International Symposium on Space Flight Dynamics - 24th ISSFD*. Laurel, MD, May 2014.

[22] Ingber, L. "Adaptive simulated annealing (ASA): Lessons learned." *Control and Cybernetics*,Vol. 25 No. 1, 1996, pp. 33-54.

[23] Ram, D. J., Sreenivas, T. H., and Subramaniam, K. G. "Parallel Simulated Annealing Algorithms." *Journal of Parallel and Distributed Computing*, Vol. 37, 1996, pp 207-212.

[24] Arshad, M., and Silaghi, M. C. "Distributed Simulated Annealing." in Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, IOS Press, series "Frontiers in Artificial Intelligence and Applications", Vol 112, 2004.

The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.