Scheduling Earth Observing Satellites with Evolutionary Algorithms

Al Globus, James Crawford, Jason Lohn and Anna Pryor

May 22, 2003

1 Abstract

We hypothesize that evolutionary algorithms can effectively schedule coordinated fleets of Earth observing satellites. The constraints are complex and the bottlenecks are not well understood, a condition where evolutionary algorithms are often effective. This is, in part, because evolutionary algorithms require only that one can represent solutions, modify solutions, and evaluate solution fitness.

To test the hypothesis we have developed a representative set of problems, produced optimization software (in Java) to solve them, and run experiments comparing techniques. This paper presents initial results of a comparison of several evolutionary and other optimization techniques; namely the genetic algorithm [5], simulated annealing [7], squeaky wheel optimization [6], and stochastic hill climbing [1]. We also compare separate satellite vs. integrated scheduling of a two satellite constellation. While the results are not definitive, tests to date suggest that simulated annealing is the best search technique and integrated scheduling is superior.

2 Introduction

A growing fleet of NASA, commercial, and foreign Earth observing satellites (EOS) uses a variety of sensing technologies for scientific, mapping, defense and commercial activities. As the number of satellites (now around 60) increases, the system as a whole will begin to approximate a sensor web. Image collection for these satellites is planned and scheduled by a variety of techniques [11], [14], [15] and others, but nearly always as separate satellites; not as an integrated sensor web. Since activities on different satellites, or even different instruments on the same satellite, are typically scheduled independently of one another, manual coordination of observations by communicating teams of mission planners is required. As sensor webs with large numbers of satellites and observation requests develop, manual coordination will no longer be possible. Schedulers that treat the entire web as a collection of resources will become necessary.

Scheduling EOS is complicated by a number of important constraints. Potin [13] lists some of these constraints as:

1. Power and thermal availability.
2. Limited imaging segments per orbit. In a given orbit, a satellite will pass over a target only once. For the sun-synchronous orbits used by most Earth observing satellites, each orbit takes about 90 minutes.
3. Revisit limitations. A target must be within sight of the satellite; and EOS satellites travel in fixed orbits. These orbits pass over any particular place on Earth at limited times so there are only a few imaging windows (and sometimes none) for a given target.
4. Time required to take each image. Most Earth observing satellites take a one dimensional image and use the spacecraft's orbital motion to sweep out the area to be imaged. Thus, the larger the image the more time is required to take it.

5. Limited on-board data storage. Images are stored in a solid state recorder (SSR) until they can be sent to the ground.

6. Ground station and communication satellite availability, especially playback opportunities. The data in the SSR can be sent to the ground either when the satellite passes over a ground station or via geosynchronous communication satellites. Ground station windows are as limited as any other target, and suitable communication satellites (mostly TDRSS) are only available when not servicing higher priority flights (e.g., shuttle or station).

7. Transition time between look angles (slewing). Some instruments are mounted on motors that can point either side-to-side (cross-track) or forward and back (along-track). In addition, some satellites can rotate to point their instruments in any direction. These are called agile satellites.

8. Cloud cover. Some sensors cannot see through clouds. Not only do clouds cover much of the Earth at any given time, but some locations are nearly always cloudy.

9. Stereo pair acquisition.

10. Coordination of multiple satellites. In a sensor web an imaging request can be satisfied by any of several satellites. Also, in many cases there is a need to image a particular area by more than one sensor, often with time constraints.

For further details of the EOS scheduling problem see [2] and [15].

We hypothesize that evolutionary algorithms can effectively schedule Earth imaging satellites, both single satellites and cooperating fleets. The constraints on such fleets are complex and the bottlenecks are not always well understood, a condition where evolutionary algorithms are often more effective than traditional techniques. Traditional techniques often require a detailed understanding of the bottlenecks, whereas evolutionary programming requires only that one can represent solutions, modify solutions, and evaluate solution fitness, not actually understand how to reason about the problem or which direction to modify solutions (no gradient information is required, although it can be used).

To test this hypothesis we have developed a (hopefully) representative set of problems and software to compare solutions generated by various evolutionary and other optimization techniques. We also present data comparing scheduling a two satellite constellation as a (small) sensor web vs. as separate systems to motivate integrated fleet scheduling.

Evolutionary and other algorithms have been applied to the EOS scheduling problem by several authors, including:

1. Sherwood et al. [15] used ASPEN, a general purpose scheduling system, to automate NASA's EO-1 satellite.

2. Potter and Gasch [14] described a clever algorithm for scheduling the Landsat 7 satellite featuring greedy search forward in time with fixup to free resources for high priority images.

3. Rao, et al. [12] reported scheduling ground station use, but not imaging activity, for a fleet of seven Indian Earth imaging satellites .

4. Lamaitre et al. [8] compared methods for sharing a satellite among multiple users. They found that fixing the fraction of the satellite devoted to each user was poor in terms of global satisfaction; whereas satisfying global criteria leads to poor performance in terms of guaranteeing a particular fraction of imaging time to each user.

5. Lamaitre's group also compared constraint programming and local search for scheduling an agile satellite [9]. They found that constraint programming is more flexible but local search performs better.

6. Wolfe and Sorensen [17] compared three algorithms, including the genetic algorithm, on the window-constrained packing problem, which is related to EOS scheduling. They found that the genetic algorithm produced the best schedules, albeit at a significant CPU cost.

7. Frank et al. [2] described plans to apply heuristic based stochastic search using the Europa [3] constraint system to EOS scheduling.

The next section describes the model problems. This is followed by a description of the optimization technique comparison software, the results of initial experiments, and future plans. Further details on the model problems, and our JavaGenes scheduling software may be found in [4].

3 Model Problems

Since our project is designed to consider the scheduling of a parameterizable generic system, not any particular spacecraft, sensor, or sensor web, it is important to develop a set of model problems that exhibit important aspects of EOS scheduling now and in the future. We have attempted to base our model's sensors and satellites on hardware currently in orbit. We have identified and begun to scope seven problems:

1. A single satellite with a single cross-track slewable instrument.
2. A two satellite constellation with satellites identical to that in problem one.
3. A single agile satellite with one instrument.
4. A single satellite with multiple instruments (one of which is slewable).
5. A sensor web of single- and multiple-instrument satellites communicating directly with the ground.
6. A sensor web of single-instrument agile satellites communicating with an in-orbit communications system based on high-data-rate lasers.
7. A sensor web with a very large number of satellites including satellites with multiple instruments. This problem presumes much cheaper and more reliable launch.

Problems 1 and 2 have been implemented. The Results section compares a number of search techniques against problem 2 with the following characteristics:

1. One week of satellite operations.
2. Two satellites in sun synchronous orbit one minute apart.
3. One identical instrument per satellite.
4. Slewing up to 48 degrees cross-track in either direction at a rate of 50 seconds/degree for each instrument.
5. 4200 imaging targets (takeImages) randomly distributed around the globe; 123 of these never come into view of either satellite.
6. 24 seconds data recording per takeImage.
7. A priority between 1 and 5 (higher priority is more important) for each takeImage.

4 EOS Scheduling by Evolutionary Algorithms and Other Optimization Techniques

There are a number of optimization (evolutionary and otherwise) algorithms in the literature. We compare a genetic algorithm (GA), simulated annealing (SA), and stochastic hill climbing (HC). In addition, we compare random and squeaky wheel (SW) transmission operators. Random transmission operators change a schedule at random (consistent with the constraints). Squeaky wheel operators examine a schedule and try to make changes that are likely to improve the schedule.

We represent a schedule as a permutation (the genotype) of the image requests (takeImages). A simple, deterministic greedy scheduler assigns resources to the requested takeImages in the order indicated by the permutation. This produces a timeline (the phenotype) with all of the scheduled takeImages, the time they are executed, and the resources used. The greedy scheduler assigns times and resources to takeImages using earliest-first scheduling heuristics while maintaining consistency with sensor availability, onboard memory (SSR) and slewing constraints.

If a takeImage cannot be scheduled without violating constraints created by scheduling takeImages from earlier in the permutation, the takeImage is left unscheduled.

Simple earliest-first scheduling starting at epoch (time = 0) had some problems, and we discovered that the algorithm works better if 'earliest-first' starts with a particular imaging window (period where the satellite is within sight of a target; most takeImages have several windows in our week-long problem) rather than at epoch. If the takeImage cannot be scheduled before the end of time, the algorithm starts at epoch and continues until the takeImage is scheduled or the initial imaging window is reached. The window within which a takeImage is scheduled is stored in memory and used by children when they generate schedules. The extra scheduling flexibility may explain why this approach works better than earliest-first starting at epoch.

Constraints are enforced by representing each resource as a timeline. Scheduling a takeImage causes each relevant resource timelines to take on appropriate values (i.e., in use for a sensor, slew motor setting, amount of SSR memory available) at different times. A takeImage is inserted at the first time examined and available in all the required resource timelines.

Search is guided by a fitness function that determines the 'goodness' of a schedule generated from a permutation. The fitness function must provide a fitness for any possible schedule, no matter how bad it is, and nearly always distinguish between any two schedules, no matter how close they are. Our fitness function is multi-objective. The objectives include:

1. Minimize the sum of the priority of the images not scheduled (takeImages). Each takeImage has a priority between 1 and 5, where the larger numbers indicate higher priority.
2. Minimize total time spent slewing (slew motors wear out).
3. Minimize the sum of the slew angles for the images taken (small slews improve image resolution).

These objectives are manipulated so that lower values are better fitness; the objectives are then combined into a weighted sum:

$$F = w_p \sum_{I_u} I_p + w_s S_t + w_a \sum_{I_s} I_a \qquad (1)$$

where F is the fitness, I_u is the set of unscheduled takeImages, I_s is the set of scheduled takeImages, I_p is the priority of a takeImage, S_t is the total time spent slewing, I_a is the slewing angle the schedule requires for a takeImage, and w_p, w_s, and w_a are weights (positive numbers).

We are now ready to describe the three search algorithms:

1. Stochastic hill climbing (HC) starts with a single randomly generated permutation. This permutation (the parent) is mutated to produce a new permutation (a child) which, if it produces a better (more fit) schedule than the parent, replaces the parent. Two cases are investigated: five restarts per run and no restarts. With no restarts, each search generates 100,000 children starting with a random permutation. In the restart case, each search consists of five subsearches of 20,000 children each; the best individual from all five searches is reported.
2. Simulated annealing (SA) is similar to HC except less fit children can replace the parent with probability $p = e^{\frac{-\triangle F}{T}}$ where $\triangle F$ is how much less fit the child is. The temperature T starts at 100 and is multiplied by 0.92 every 1000 children (100,000 children are generated per run).
3. The genetic algorithm (GA) seeks to mimic the natural evolution of populations of organisms and there are many variants. Our GA employs the following algorithm:
   (a) Generate a population of 100 random permutations
   (b) Calculate the fitness of each permutation
   (c) Repeat
      i. Randomly select parent permutations with a bias towards better fitness
      ii. Produce child permutations from the parents with:

A. crossover that combines parts of two parents into a child, or

B. mutation that modifies a single parent

iii. Calculate the fitness of the child

iv. Randomly replace individuals of less fitness in the population with the children

(d) Until 100,000 children have been produced

The search for a good schedule starts with one or more random permutation (the initial parents) and uses mutation and crossover operators to create children from parents. This paper compares four mutation operators and one crossover operator. The mutation operators are:

1. Random swap. Two permutation locations are chosen at random and the takeImages are swapped. Swaps are executed 1-9 times per mutation. A single random swap is called order-based mutation [16].

2. Squeaky swap. This is the same as random swap except that the takeImages to swap are chosen more carefully. Specifically, a tournament of size 10, 20, 50, 100, 200, or 500 selects both takeImages. One takeImage that 'should' be moved forward in the permutation is chosen. The winning takeImage is (in this order):

   (a) unscheduled rather than scheduled

   (b) higher priority

   (c) later in the permutation

   The other takeImage is chosen assuming it should be moved back in the permutation. This tournament winner has the opposite characteristics. Although the takeImages to swap are chosen because one 'should' move forward in the permutation and the other 'should' move back, this is not enforced. Experiment determined that the desired direction of the swap did not actually occur nearly as often as expected, occasionally less than half the time!

3. Placed squeaky swap. Here the direction is enforced. A separate tournament (of size 10, 20, 50, 100, or 200) is conducted for each takeImage. The takeImage to move forward is forced to be in the last half of the permutation. The takeImage to move back is then forced to be at least half way towards the front.

4. Cut and rearrange. The permutation is cut into 1-5 pieces and these are put back together in a random order. This is similar to the cut-set based operators used in the traveling salesman problem community.

The crossover operator is only used in the genetic algorithm. The operator is Syswerda and Palmucci's position-based crossover [16]. Roughly half of the permutation positions are chosen at random (50% probability per position). These positions are copied from the father to the child. The remaining takeImage numbers fill in the other positions in the order they appear in the mother.

In many cases several different transmission operators and/or the same kind of operator with different sized tournaments, number of swaps, or cuts were used. In these cases, each child was produced by a randomly chosen transmission operator.

5 Results

A number of search technique/transmission operator pairs were compared. Each combination was repeated 94 times to get statistically significant results. The resulting distributions were spot checked for a gaussian distribution to insure the Student's T-test is valid. In each trial, evolution produced 100,000 children.

A quantitative comparison of search techniques and transmission operators (various forms of mutation and crossover) can be found in table I. The techniques at the top of the table produce the best schedules, the techniques at the bottom the worst. A few observations:

1. Simulated annealing is clearly the best search technique. It is not surprising the SA beats

search algorithm	transmission operators	fitness (equation 1)	priority ($w_p \sum_{I_u} I_p$)	takeImage (I_u)
SA	1-9 swap	2171	1873	1199
SA	1 swap	2354	2077	1295
HC 5 restarts	1-9 swaps	2539	2287	1415
HC 5 restarts	1 swap	2564	2313	1429
HC 0 restarts	1 swap	2575	2327	1436
SA	1 squeaky swap	2772	2527	1615
SA	1 placed squeaky swap	2814	2559	1579
HC	1 squeaky swap	2868	2625	1623
GA population = 100	crossover and 1 swap	3007	2759	1558
GA population = 100	1-5 cut and rearranges	3008	2754	1526
SA	1-5 cut and rearranges	3012	2737	1439

Table 1: Comparison of search techniques. Search-technique/transmission operator pairs ordered by mean fitness. Techniques are ordered by fitness (low values are better schedules for all measures). Priority is the sum of the priority of all unscheduled tasks. TakeImage is the number of unscheduled takeImages. All data are the mean of 94 searches. Values are rounded down to the next lowest whole number. All differences are statistically significant (as measured by Student's T-Test) except for fitness: HC with 0 and 5 restarts with 1 swap, and the worst three; priority: only the worst three; and several of the takeImage comparisons.

HC, since HC is clearly vulnerable to local minima. To understand why SA and HC beat GA, consider the building blocks in the permutation. These may be thought of as sets of takeImages in a particular order that leads to good partial schedules. Moving an arbitrary takeImage before a building block can easily disrupt it by making some of the takeImages unschedulable; or worse, causing one of the takeImages in the building block to be scheduled in another window further disrupting the building block. Since good building blocks are thought to be essential to GA performance [5], GA does poorly.

2. Random swap mutations beat the smarter 'squeaky' mutation where the takeImages to swap are chosen more carefully (a counter intuitive result). This may be, in part, because the squeaky operators limit the possible moves an algorithm may take. This can create additional local minima that the search then falls into.

3. Multiple swaps are better than a single swap, possibly because some moves are impossible with a single swap.

4. Ordering techniques by priority or takeImage rather than fitness doesn't make any difference for the best techniques, and much of the difference that does occur is not statistically significant.

5. The cut and rearrange operators do very poorly. Cut and rearrange works well for the traveling salesman problem because moving contiguous chunks of the permutation relative to each other does not change the partial fitness of the chunk. In permutation driven scheduling, however, reversing the order of two contiguous chunks can cause very large changes in the schedule.

These observations should be considered preliminary rather than definitive. First of all, this is a single problem and results may vary when a larger range of the model problems are addressed. Second, the squeaky algorithms can stand improvement and may someday outperform the random operators. Nonetheless, if these results stand up, there are some important implications.

1. Simulated annealing requires less memory than

the genetic algorithm and does not require crossover operators or a population, making it better performing, more efficient, and easier to implement.

2. Random swaps out perform the 'smarter' squeaky swaps, making random swaps better performing, faster, and easier to implement.

3. One should allow multiple random swaps, in spite of the minor increase in code complexity.

Figure 1 shows the evolutionary history of the best individuals for the best schedules evolved by simulated annealing (SA), hill climbing (HC), and the genetic algorithm (GA) using the the one random swap mutation operator. Notice that although simulated annealing wins in the end, it trails GA until about generation 50 and trails HC until about generation 70. SA seems to be doing a better job of finding and then exploiting a deep minimum. Notice also that all three techniques are still improving the schedule at the end of the run, suggesting that additional evolution (more than 100,000 children) would be rewarded with better schedules.

One unexpected property of the schedules generated was the slewing. Specifically, in order to minimize total slewing time (S_t from equation 1) the schedules tended to place takeImages such that the instrument is slewed to extremes (see figure 2); which will generate relatively low resolution images. This could perhaps be improved if the fitness function gave more weight to minimizing the sum of the slews or if the instruments slewed faster (which would also be more realistic).

A second experiment compared GA with one swap operator on two problems: in the first, each satellite was randomly assigned half of 4,200 takeImages; in the second, either satellite was allowed to execute any takeImage (see table 2). As might be expected, the case where any satellite could take any image produced superior schedules. Specifically, the shared case was able to take about 28% more images, the priority measure improved 40%, and fitness 35%. This suggests that integrated fleet scheduling is much better than separately scheduling each satellite or sensor.

6 Future Work

Future work will be focused on expanding table I to include more problems and techniques. Specifically, we intend to add:

1. Additional model problems.

2. A duty cycle constraint. This constraint requires that an instrument is not used for more than u seconds in any t second time period.

3. Improved squeaky operators; in particular, shifting a high priority, unscheduled takeImage forward, rather than swapping with a scheduled, low priority takeImage.

4. Swap operators where the number of swaps is a probabilistic function of the number of children that have been produced. As evolution proceeds, the number of swaps is reduced. This encourages large steps in the beginning of evolution and smaller refinement steps near the end.

5. Transmission operator evolution; where transmission operators that have done well early in evolution are more likely to be used.

6. Additional forms of local search.

7. HBSS (Heuristic Biased Stochastic Search) with contention based heuristics similar to those proposed in [2].

8. A multi-objective co-evolution genetic algorithm [10]. The present fitness function depends on somewhat arbitrary weights to turn multiple objectives into a single objective for fitness comparisons. A true multi-objective approach might generate better schedules.

9. Changing the fitness function to normalize the sum of the amount of slew for each scheduled takeImage by the number of scheduled takeImages. At present, this objective *increases* as more takeImages are scheduled, penalizing what we want to do!

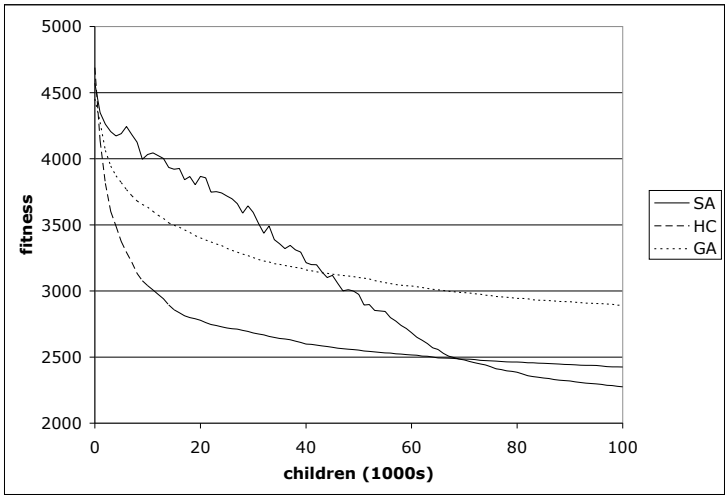

Figure 1: A comparison of the evolutionary history of simulated annealing, hill climbing, and the genetic algorithm. Lower fitness values indicate better schedules.

problem	fitness (equation 1)	priority ($w_p \sum_{I_u} I_p$)	takeImage (I_u)
shared	2171	1873	1199
separate	3346	3096	1657

Table 2: Comparison of shared target vs separate targets for a two satellite constellation using GA with only single swap mutation and crossover. All comparisons are statistically significant. The shared case is 25-40% better depending on the measure used for comparison.

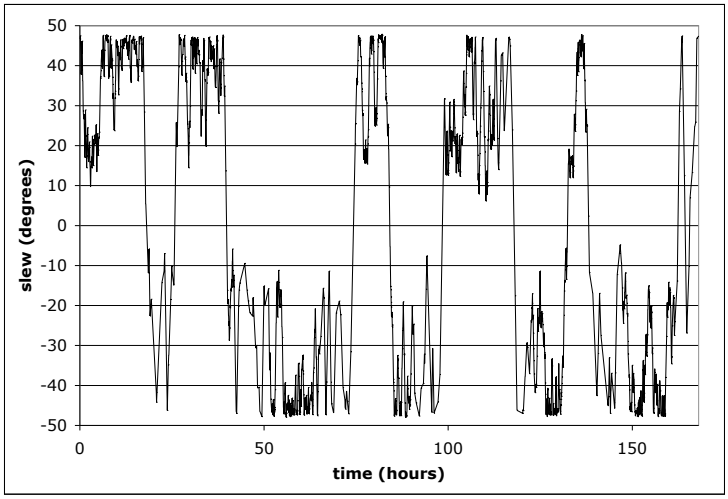

Figure 2: The slew history for one satellite in the best schedule generated. The horizontal axis is time; a total of one week. The vertical axis is the amount of cross-track slew necessary to execute the scheduled takeImages for this satellite. Note the preference for extreme slews. The extreme slews apparently minimize the total slewing time sufficiently to overcome the fitness pressure towards small slews.

7 Summary

Earth imaging satellite constellation scheduling is a complex task with many variables and interacting constraints. We hypothesize that evolutionary programming can solve the EOS scheduling problem effectively and have begun to test various evolutionary search techniques and transmission operators. Simulated annealing with 1-9 random swaps performed the best of those techniques we have tested on a single two-satellite problem. We have also shown that scheduling a small fleet as a combined resource outperforms separate scheduling for each satellite by about 25-40%.

8 Acknowledgements

This work was funded by NASA's Computing, Information, & Communications Technology Program, Advanced Information Systems Technology Program (contract AIST-0042), and by the Intelligent Systems Program. Thanks to Alex Herz, Orbit Logic, Inc. and Jerald Arp, PhD., Manager of Technical Support, Space Imaging LLC for information regarding current EOS systems. Thanks also to Bonnie Klein for reviewing this paper and to Jennifer Dungan, Jeremy Frank, Robert Morris and David Smith for many helpful discussions. Finally, thanks to the developers of the excellent Colt open source libraries for high performance scientific and technical computing in Java (http://hoschek.home.cern.ch/hoschek/colt).

References

[1] Shumeet Baluja, "An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics," Carnegie Mellon University technical report CMU-CS-95-193, September 1995.

[2] Jeremy Frank, Ari Jonsson, Robert Morris, and David Smith, "Planning and Scheduling for Fleets of Earth Observing Satellites," *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space 2002*, Montreal, 18-22 June 2002.

[3] Jeremy Frank and Ari Jonsson, "Constraint-based Attribute and Interval Planning," to appear in the *Journal of Constraints, Special Issue on Constraints and Planning*.

[4] Al Globus, James Crawford, Jason Lohn, and Robert Morris, "Scheduling Earth Observing Fleets Using Evolutionary Algorithms: Problem Description and Approach," *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, NASA, Houston, Texas, 27-29 October, 2002.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[6] D. E. Joslin and D. P. Clements, "Squeaky Wheel Optimization," *Journal of Artificial Intelligence Research*, volume 10, pages 353-373, 1999.

[7] S. Kirkpatrick, C. D. Gelatt, and M.P. AndVecchi, "Optimization by Simulated Annealing," *Science*, volume 220, number 4598, pages 671-680, May 1983.

[8] M. Lamaitre, G. Verfaillie, and N. Bataille, "Sharing the Use of a Satellite: an Overview of Methods," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998

[9] M. Lamaitre, G. Verfaillie, J. Frank, J. Lachiver, and N. Bataille, "How to Manage the New Generation of Agile Earth Observation Satellites," *SpaceOps 2000*, sponsored by the Centre National dEstudes Spatialles (CNES) of France, 2000.

[10] J.D. Lohn, G.L. Haith, S.P. Colombano, D. Stassinopoulos, "A Comparison of Dynamic Fitness Schedules for Evolutionary Design of Amplifiers," *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA, IEEE Computer Society Press, pages 87-92, 1999.

[11] H. Muraoka, R. H. Cohen, T. Ohno, and N. Doi, "Aster Observation Scheduling Algorithm," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998.

[12] J. D. Rao, P. Soma, G. S. Padmashree, G.S. "Multi-Satellite Scheduling System for LEO Satellite Operations," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998.

[13] P. Potin, "End-To-End Planning Approach for Earth Observation Mission Exploitation," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998.

[14] W. Potter and J. Gasch, "A Photo Album of Earth: Scheduling Landsat 7 Mission Daily Activities," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998.

[15] R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, and A. Fukunaga, "Using ASPEN to Automate EO-1 Activity Planning," *Proceedings of the 1998 IEEE Aerospace Conference*, Aspen, Colorado, March 1998.

[16] G. Syswerda and J. Palmucci, "The Application of Genetic Algorithms to Resource Scheduling," *Proceedings of the Fourth International Conference on Genetic Algorithms*, University of California, San Diego, Richard K. Belew and Lashon B. Booker, editors, pages 502-508, 13-16 July 1991.

[17] W. J. Wolfe and S. E. Sorensen, "Three Scheduling Algorithms Applied to the Earth Observing Systems Domain," *Management Science*, volume 46, number 1, pages 148-168, January 2000.

[18] Y. Yamaguchi, T. Kawakami, A. B. Kahle, M. Pniel, and H. Tsu, H., "Aster Mission Planning and Operations Concept," *SpaceOps 1998*, Tokyo, Japan, 1-5 June 1998.