Rapid Scheduling of Multi-tracking Sensors for a Responsive Satellite Surveillance Network

Dick Stottler[1], John L.Mohammed, Ph.D.
Stottler Henke Associates, Inc.; 951 Mariner's Island Blvd., STE. 360, San Mateo, CA. 94404

Satellites currently play a critical role in military operations, providing functions such as targeting, surveillance, communications and navigation. The Satellite Surveillance Network (SSN) is a small network of geographically distributed sites that provides tracking for all orbiting objects, and maintaining a Space Catalog of their orbits. The SSN currently tracks over 16,000 objects. For each object there is an associated priority (1 through 5) and a requisite number of tracks to obtain per day. Most SSN sites consist of sensors that can either track 1 object at a time (such as mechanically tracked parabolic dish radars) or that have a fixed field of view and can track many objects at a time such as phased array radars. But there are also interesting future sensor systems that can track multiple objects at a time and have a movable field of view. Scheduling of the observations of these types of assets is the emphasis of this paper. Examples include the Space Surveillance Telescope (SST) developed by DARPA and MIT/LL which is wide angle searching and tracking telescope and airborne and space-based phased array radars. The key scheduling task is to determine where the sensor should be aimed (i.e., what volume of the sky should be viewed) over time in order to ensure that they can obtain as many of the tracks with which they are tasked as possible, and that all the highest priority tracks are obtained. This is a difficult optimization problem in a continuous three-dimensional space (the coordinates are the azimuth and elevation at which to aim the sensor, and the time at which it should be aimed there). The objective is to maximize the number of targets visible at any given time so as to satisfy as many track requests as possible, while conforming to the requirements concerning priority, number of tracks and number of observations per track. We describe a new approach to this scheduling task that employs a novel data-driven approach to discretizing the search space. This approach results in rapid search of a tractable search space containing high-quality options that maximize the number of tracks that can be scheduled. Our approach defines techniques for clustering objects whose visibility sets intersect. These intersections are discrete subsets of the search space which are then treated as resources in a constraint-based scheduling algorithm. The scheduling algorithm ensures that the final selection of aim-points contains fields of view that do not overlap in time, and that satisfy as many track requests as possible, taking priority into account.

[1] Job Title: President; Department Name: N/A; Street Address/Mail Stop: 951 Mariner's Island Blvd., STE. 360, San Mateo, CA. 94404; AIAA Member Grade: Regular New

Results of testing the algorithm with 100, 250, 1,000 and 10,000 objects with randomly assigned priorities are presented. In each case, the scheduler rapidly generated a schedule that satisfied every track request. On a laptop personal computer, the maximum total time to compute a schedule for 16,960 track requests for 10,000 objects was 15 minutes 34 seconds. Incremental changes were also fast. The maximum time to add 30 track requests for 5 objects to a schedule for 10,000 objects was 2.9 seconds.

I OVERVIEW OF SOLUTION

In general, scheduling any set of resources is an NP-complete problem. Thus, algorithms to create schedules often become extremely slow as the problem size grows, and none are guaranteed to produce optimal schedules. In this situation, the problem is complicated by the fact that the search space is continuous and three-dimensional. If one is not careful how the three-dimensional space is carved up into discrete chunks, one can make the discrete space to search too large, making search intractable, or limit the scheduler to choosing among poorly defined options, resulting in poor quality schedules.

We developed a novel, data-driven approach to discretizing the search space that results in rapid search of a tractable search space containing high-quality options that maximize the number of tracks that can be scheduled. We identified "visibility sets" in the azimuth-elevation parameter space as a key representation that enables us to identify the most efficient aim-points for the sensors, and thus discretize the search space, enabling the schedule to be computed by a constraint-based scheduling system.

A. Multi-Track Sensor Scheduling

Objects in orbit move across the sky, periodically entering and leaving the potential field of view for a specific sensor. The potential contacts are especially short and relatively rare for objects in low-earth orbit. If the sensor's field of view could encompass the entire sky visible from its location, then scheduling would not be an issue. However, the sensor's field of view at any given time is limited to a particular subset of the visible sky, which can be characterized by a range of azimuth and elevation values. The field of view can also be characterized by an aim-point that specifies the center of the azimuth and elevation ranges.

A specific object can be tracked if it passes through the sensor's field of view for a sufficient length of time to acquire the required number of observations. We can plot the course of the object for a contact in the parameter space of the polar azimuth and elevation coordinates from the perspective of the sensor. In this space the field of view of the sensor is represented by a square indicating the range of azimuths and elevations that are visible. We can ensure that the object can be tracked by placing the visibility square at any point for which a sufficiently long continuous segment of the object's track is contained within the square (see Figure 1). For each contact window of each object we can define the "visibility set" as the set of points that describe the centers of visibility squares that satisfy the tracking condition. This will in general be a volume surrounding the track in the 3-dimensional space of azimuth, elevation and time (see Figure 2). Each member of the set would have an associated time range determined by the period of time that the object intersects the square that the point represents.

For two objects to be visible within the same field of view, their visibility sets must have a non-empty intersection, and the sensor's aim-point must be within that intersection. To schedule the sensor, we must select an appropriate allocation of aim-points as a function of time. To maximize the number of objects tracked, these aim-point selections should be in the intersection sets for the maximum number of objects. In Figure 2 two times when several visibility sets intersect are indicated by square planes.

We developed algorithms to determine the most effective selection of aim-points, based on techniques for clustering objects that have intersecting visibility sets, and on histogram techniques for determining the fields of view that cover the largest number of the most important objects. These algorithms generate sets of potential aim-points, which are then treated as resources by a scheduling algorithm (see Figure 3). The scheduling algorithm

ensures that the final selection of aim-points contains fields of view that do not overlap in time, and that satisfy the requirements for the number and spacing of tracks for as many objects as possible, taking into account each object's priority.

The scheduling algorithm first allocates aim-points for those objects that have the fewest opportunities for tracking (in comparison to the number of tracks needed), independent of priority. Then, if the resulting schedule fails to adequately track any higher priority objects, the algorithm attempts to incorporate the aim-points needed for the higher priority object in two ways: 1) by removing aim-points that cover lower priority objects and substituting ones that provide the missing tracks for the higher priority objects, or 2) by shifting existing aim-points slightly so that they can provide the necessary missing tracks, at the possible expense of losing tracks for lower priority objects. In fact, the second approach is attempted first, because shifting the aim-point slightly can sometimes incorporate the missing tracks without losing any already scheduled tracks.

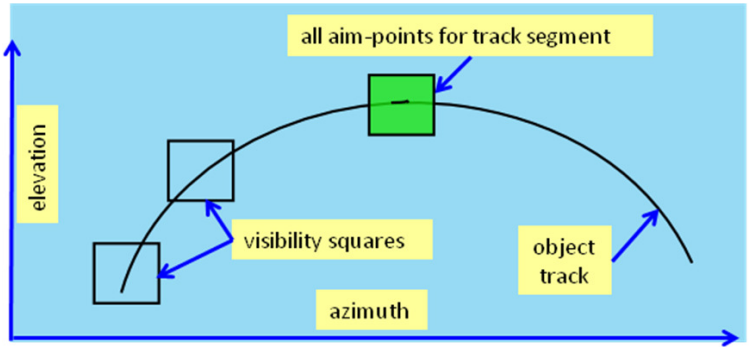

Figure 1. Object track and visibility squares in azimuth, elevation parameter space.

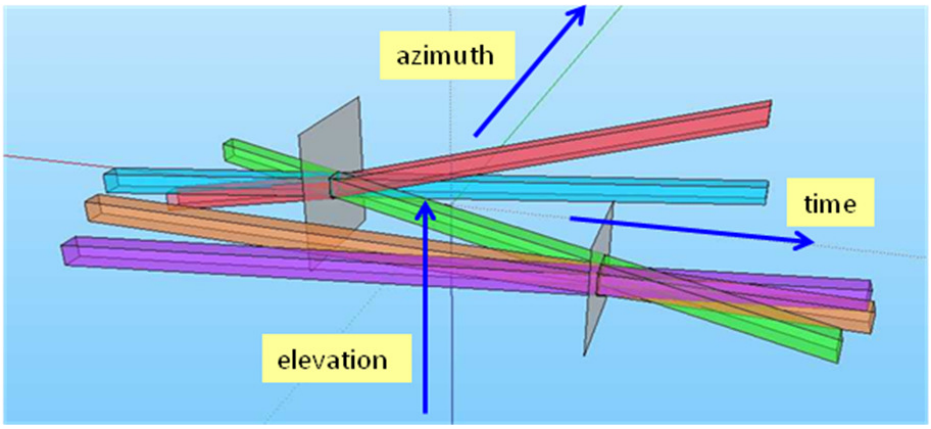

Figure 2. Visibility set volumes for a series of tracks. Each "tube" in the image represents possible aim-points within which a particular object is visible at any given time. Places where the visibility sets intersect (such as the places indicated in the image by planes) represent times and aim-points when several objects are visible simultaneously.

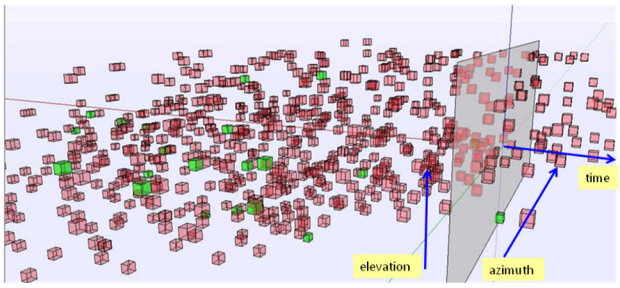

Figure 3. Visibility set clusters. In this figure each cube represents a set of aim-points for which several objects are visible at the same time (the grey square indicates the azimuth-elevation plane, and time runs orthogonal to that plane). The green cubes represent visibility sets from which an aim-point has been selected for inclusion in the schedule, whereas the pink cubes represent visibility sets not included in the schedule.

II. THE SCHEDULING PROTOTYPE

We implemented the visibility set clustering algorithm as a Java™[2] program. The program accepts as input a set of targets. Each target has a priority, a number of tracks requested, and the number of seconds on which the sensor must dwell on the object to receive the appropriate number of observations for each track (Fig. 4.). The name of each target also identifies a text file containing the ephemerides, in the polar azimuth elevation space for accesses of the target from a specific ground site. These were generated in AGI's Satellite ToolKit (STK™[3]) and are in the form of one of their reports (

Figure). The program parses these text files to obtain the ephemerides for each possible contact with each target.

The prototype constructs a visibility set for each contact with each target then employs a clustering algorithm to compute visibility sets for the intersections of these visibility sets. The clustering algorithm attempts to determine the times and aim-points for which the largest number of satellites are visible simultaneously. These visibility sets are employed as resources in a simple constraint-based scheduling algorithm also implemented in Java™.

tle-00694,1,5,30
tle-00733,1,5,30
tle-00877,2,5,30
tle-02802,2,5,30
tle-03230,2,5,30
tle-03597,2,5,30
tle-03669,2,5,30
tle-03835,3,4,30
tle-04327,2,5,30
tle-04814,2,5,30

Figure 4. Track Request Input to the Prototype. Each Object is identified by its two-line element name, and associates with a priority (1 to 5), a number of requested tracks, and a number of seconds needed to obtain an appropriate number of observations.

[2] Java™ is a trademark of Sun Microsystems, Inc.
[3] STK™ is a trademark of Analytical Graphics, Inc.

```
                                                          16 Aug 2009 14:27:38
Facility-Facility1-To-Satellite-tle-00694:  Inview Azimuth, Elevation, & Range


Facility1-To-tle-00694
----------------------
                          Time (UTCG)          Azimuth (deg)    Elevation (deg)
                  ------------------------    -------------    ---------------
                  16 Aug 2009 12:00:00.000          177.790            34.624
                  16 Aug 2009 12:00:10.000          174.563            34.324
                  16 Aug 2009 12:00:20.000          171.376            33.923
                  16 Aug 2009 12:00:30.000          168.246            33.424
```

Figure 5. Ephemerides file. For each target there is a file like this one that specifies the location of the satellite as a function of time in the polar coordinates of the sensor's field of view.

Two versions of the visibility set computation were implemented. The first version made an unreasonable assumption regarding the shape of the beam that would be created by the sensor. To simplify computations, the initial version of the prototype assumes that the beam shape is bounded by planes of constant azimuth or elevation (see Fig. 6). This implies that at the higher elevations the beam cross-section is smaller and more trapezoidal, becoming triangular when the uppermost point is at ninety degrees. This is unreasonable from the standpoint of the physics of the sensor system. The impact of this assumption on the schedule is that the algorithm "prefers" contacts at lower elevations.

A second version of the computation was then implemented in which the shape is independent of elevation. This makes computation of visibility set intersections more complex, as it is necessary to compute intersections of convex great-circle spherical polygons. However, we developed a rapid and numerically stable technique for computing these intersections, resulting in an efficient algorithm for computing visibility sets with a far more physically reasonable assumption regarding beam shape (see Fig. 7).

Note that sensors are generally restricted to a field of view that is less than the maximum possible. The prototype can restrict the overall field of view to meet various constraints. However, we only tested it with the assumption that its field of view is 360 degrees in azimuth, and from 5 to 90 degrees in elevation.

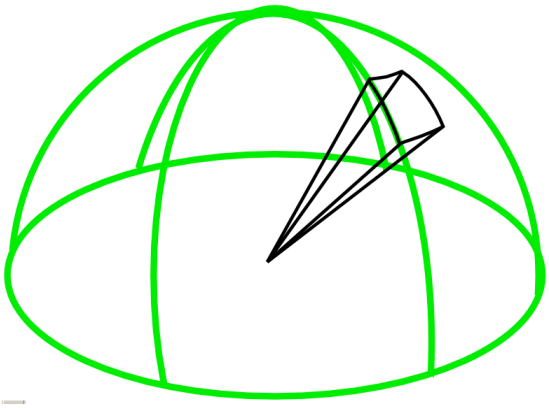

Figure 6. Beam-shape determined by planes of azimuth and elevation.

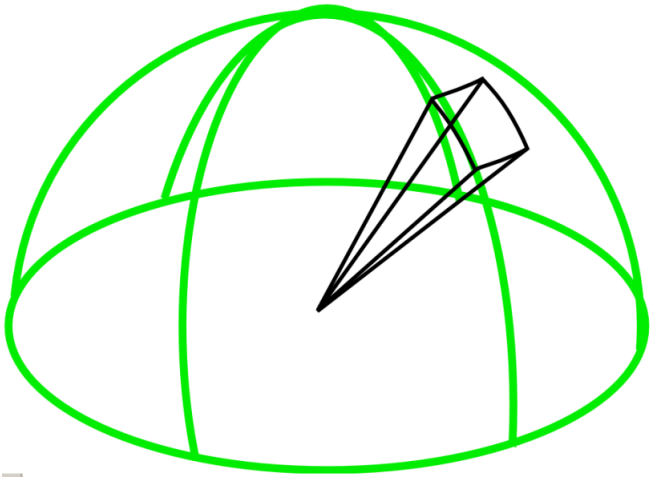

Figure 7. Beam shape invariant to azimuth and elevation.

Our purpose in constructing and testing the prototype was to demonstrate that it is possible to rapidly generate valid schedules. To test it we obtained two-line elements (TLEs) from a public website, the Celestrak NORAD current Space Track data access (http://celestrak.com/NORAD/elements/).

However, for this effort we had to manually construct a set of track requests that randomly assigned priorities to the satellites. Further, to obtain ephemerides we had to load the TLEs into STK, manually request the generation of an Azimuth Elevation Range (AER) report for each satellite, and manually save the reports to text files. Thus, it was not an efficient use of time to perform this preparation for a large number of satellites. We therefore, for the large number of satellite test cases, made some simplifications in the data. Specifically, although the priority is independent of the number of tracks per day needed, for the automatically generated test data, the number of tracks per day for each satellite was simply generated based on its priority.

Initially, we downloaded the "100 Brightest" dataset from that website, and tested the scheduler using this small dataset. Later we manually prepared an additional 150 satellites, including 96 elements of the Iridium debris, enabling us to test the scheduler with 250 satellites. To test the ability of the scheduler to scale up to more realistically sized datasets, we then programmatically replicated the 250 satellites, distributing the copies of each satellites evenly in time. Replicating each satellite four times gave us a dataset with 1000 satellites, and replicating each satellite 40 times created a dataset with 10,000 satellites.

For the tests with 100 satellites, we requested that the scheduler insert a track at every opportunity for every satellite. That is, the scheduler had to satisfy one track request for each satellite for every contact window. Furthermore, we required that the satellites be visible for 30 seconds worth of observations for each track.

For the tests with 250, 1000 and 10,000 satellites, the number of track requests for each satellite was a function of the satellite's priority, and only 10 seconds of observations were required per track. Higher priority satellites had to have more track requests satisfied, and lower priority satellites required fewer. Table 1 shows the relationship between satellite priority and number of track requests for these automatically generated test data.

Table 2 shows the distribution of priorities among the satellites. Priorities were assigned randomly, with the number of satellites assigned each priority being inversely proportional to the priority. Table 1 shows the number of track requests as a function of priority, as well as the total number of track requests made for each test.

Table 1. Number of track requests as a function of satellite priority.

Priority	Number of track requests
1	Every opportunity
2	4
3	3
4	2
5	1

Table 2. Distribution of priorities among satellites.

Priority	Dataset size			
	100	**250**	**1,000**	**10,000**
1	5	7	28	280
2	9	11	44	440
3	23	30	120	1,200
4	30	55	220	2,200
5	33	147	588	5,880

Table 1. Distribution of Track requests among satellites.

Priority	Dataset size			
	100	**250**	**1,000**	**10,000**
1	25	34	136	1360
2	42	49	172	1720
3	111	143	360	3600
4	140	258	440	4400
5	166	724	588	5880
Total	484	1208	1696	16960

In each case we created a schedule for one 24-hour period, and used the computer's timer to measure the amount of time required to create the visibility sets, to create the schedule, and the total time. In each case we ran the tests four times and computed the minimum, maximum and mean times taken. The tests were run on a laptop computer with a clock speed of 789 Mhz, and 2 GB of RAM, running the Windows XP™ operating system.

In each case the scheduler was able satisfy all the track requests. Table 4 shows the time it took to compute and cluster the visibility sets. Table 5 shows the time requires to create the aim-point schedule, and Table 6 shows the total times. All times are in milliseconds. Figure 8 shows a log-log plot of the mean times to build the schedules.

Table 4. Time to compute visibility set clusters.

Dataset Size	Test 1	Test 2	Test 3	Test 4	Minimum	Maximum	Mean
100	1547	1610	1453	1156	1156	1610	1442
250	3438	3500	3422	3438	3422	3500	3450
1000	15969	15984	16485	15938	15938	16485	16094
10,000	816953	778015	791016	773203	773203	816953	789797

Table 5. Time to compute aim-point schedules.

Dataset Size	Test 1	Test 2	Test 3	Test 4	Minimum	Maximum	Mean
100	312	359	313	172	172	359	289
250	750	844	766	750	750	844	778
1000	1969	2016	2140	1953	1953	2140	2020
10,000	104078	112641	112062	112281	104078	112641	110266

Table 2. Total time to build schedule.

Dataset Size	Test 1	Test 2	Test 3	Test 4	Minimum	Maximum	Mean
100	1859	1969	1766	1328	1328	1969	1731
250	4188	4344	4188	4188	4188	4344	4227
1000	17938	18000	18625	17891	17891	18625	18114
10,000	921031	890656	903078	885484	885484	921031	900062

For the 250, 1000 and 10,000 satellite datasets, we also requested that the scheduler update the priority of some of the satellites, thus adding additional track requests for those satellites. For the 250 satellite dataset, one additional track was requested for one satellite. For the 1000 and 10,000 satellite datasets, 5 satellites had their priority increased from 5 (the lowest) to 1 (the highest), thus increasing the number of track requests for each satellite from 1 to 6 (every opportunity). In each case, the schedule was rapidly updated to include all the newly requested tracks while still satisfying all previously requested tracks. Table 7 shows the time the scheduler required to perform these updates.

Table 3. Time to update schedule with additional requests.

Dataset Size	Test 1	Test 2	Test 3	Test 4	Minimum	Maximum	Mean
250	0	0	0	16	0	16	4
1000	47	47	63	63	47	63	55
10,000	235	375	360	359	235	375	332

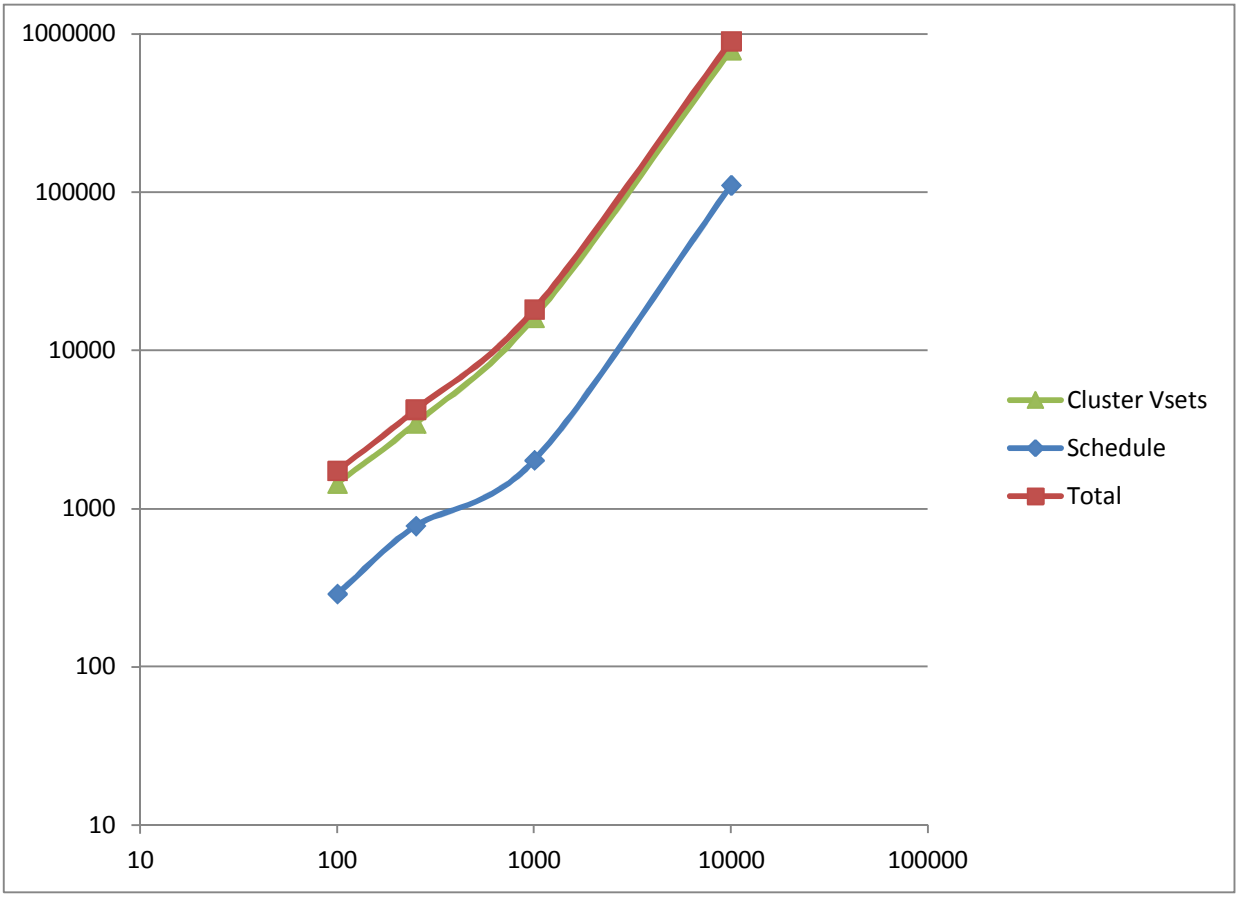

Figure 8. Log-Log plot of mean times in milliseconds to build complete aim-point schedules vs. dataset size

The scheduler is able to satisfy all the requests because it finds aim-points that maximize the number satellites that can be tracked simultaneously. Table 8 illustrates the multiplicity of tracked satellites by comparing the total number of track requests with the total number of aim-points in the schedule.

Table 4. Track multiplicity.

Dataset Size	Total # Track requests	Total # aim-points
100	484	326
250	1207	641
1000	1696	553
10,000	16960	2932

IV. RELATED WORK

There has been progress on automating the planning and scheduling of unmanned space missions. These are relevant because of the similarity of the constraints (loss of signal between orbiting objects and ground locations). Developers at NASA JPL developed an automated planning and scheduling system named ASPEN. ASPEN can generate a sequence of low-level spacecraft commands from a high-level goal, and manage the scheduling of spacecraft resources. ASPEN is unique among planners/schedulers specialized to unmanned spacecraft in that it is modular and so can be adapted to multiple missions[1]. Unfortunately its typical scheduling paradigm is to develop an initial schedule quickly then iteratively repair and optimize it. This does not tend to work well in a very full schedule with many tight constraints, especially when there is very little room within most view periods to move the sensing event around.

The scheduling system here described is closely related to the three following areas of application for which automatic scheduling prototypes have been built: the Air Force Satellite Control Network (AFSCN), NASA's Deep Space Network (DSN) and Earth Observing Satellites (EOA) scheduling. The AFSCN coordinates communications to more than 100 satellites via nine ground stations positioned around the world. Customers request an antenna at a ground station for a specific time window along with possible alternative slots. Typically, 500 requests a day result in more than 100 conflicts[2]. The DSN is a multi-national collection of ground-based radio antennas responsible for maintaining communications with research satellites and deep space probes. The DSN maintains 16 antennas that provide tracking, navigation, and data transmission services, among others[3,4]. Earth Observing Satellites (EOA) employ advanced sensing technology for scientific, mapping, defense and commercial activities[5,6].

There are four types of scheduling algorithms applied in these areas: "exact" methods, iterative repair algorithms, heuristic search, and stochastic search. We next describe these techniques. The exact scheduling techniques pose the scheduling problem as a Mixed Integer Programming (MIP) problem which is then solved by linear programming techniques like Branch and Bound, Branch and Cut, etc. For example, Gooley developed a two phase algorithm to solve the AFSCN scheduling problem[7]: in the first phase the low altitude requests are scheduled using MIP. In the second phase, the high altitude requests are inserted in the schedule without rescheduling any of the low altitude requests.

Iterative repair algorithms generate some initial schedule and then iteratively try to fix existing conflicts in the schedule. For example, Clement and Johnston[4] use ASPEN[1] to model a DSN application. ASPEN is a local search, heuristic iterative repair planner/scheduler: it takes an initial schedule and iteratively adds, removes, reschedules and refines tasks in the schedule for as long as it is invoked, until all conflicts are resolved, or until a perfect utility is obtained. DSN heuristics include: (i) half of the time choose the conflict to repair based on a default measure of badness for the conflict type; otherwise, choose the earliest occurring; (ii) for violated constraints choose the move

repair method 70% of the time; abstract and detail 30% of the time; (iii) choose antennas within view ten times more often than those not in view.

The heuristic search methods apply some domain dependent heuristic to determine the order in which a task request should be scheduled. Once the order is determined, tasks are scheduled usually using some greedy scheduler. For instance, in the AFSCN domain a typical heuristic tries to schedule first low altitude requests before high altitude requests, sorting requests in decreasing order of the ratio of duration of the request to the average length of its time windows; ties are broken based on the number of alternative resources specified (fewer alternatives scheduled first). A greedy scheduler will consider the tasks in the given sorted order, and assign the first available resource to the task. If such resources are not available then the task is left unscheduled[2].

The stochastic scheduling techniques determine the order in which tasks are to be scheduled by doing a search on the space of such orders. A cost/utility function is used to guide the search (e.g., minimize the number of conflicts in the resulting schedule). Different search techniques can be used. For example, a stochastic Hill Climbing technique starts with a single random order of tasks. This order (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. Another common technique is Genetic Algorithms (GA) where a population of solutions is maintained. At each step of the algorithm, a pair of parent solutions is selected, and a crossover operator is used to generate a single child solution, which then replaces the worst solution in the population. GA algorithms have been proposed to solve the EOA scheduling problem[6] and the AFSCN scheduling problem[2].

The AFSCN, DNS and EOA applications all have in common the need to schedule the use of communication antennas during available view periods. They differ in the type of task constraints and resource requirements they will require, and so not all scheduling algorithms work properly for the three applications. For example, for the AFSCN, local repair techniques have been found less effective than approaches that make more moves at once[2]. In the EOA domain Evolutionary Algorithms have been found more effective than other scheduling techniques, mainly because in EOA applications constraints are complex and the bottlenecks are not well understood, making it hard to come up with effective domain scheduling heuristics[6]. In the DNS application, local heuristic search combined with systematic search has been proven effective (yet possibly computationally expensive)[4].

Most of the above techniques suffer from the same problem: finding a suboptimal scheduling allocation. This is in part because the decisions to resolve a conflict, or to define the order in which tasks should be allocated, use local information (a particular task's requirements) rather than global information about the interactions among tasks and resources. The system described here employs an architecture where the algorithms to make the main scheduling decisions (e.g., which task to schedule first, which resource and time window to select, which conflict to solve next, how to solve this conflict) can be easily customized to fix the particular needs of the domain of application. For instance, the different types of scheduling techniques described above can be implemented using the proposed architecture.

James Miller gives a good description of the overall problem of allocating tracks to all the sensors in the SSN, and the factors that must be considered when deciding satellite priorities and the quality of the schedules[8]. However, he does not deal with the issue of second-by-second scheduling of the sensors to satisfy their allotted tracks, noting that each site must be responsible for its own scheduling. Abbot and Wallace give a detailed account of the requirements and techniques for tracking objects in geosynchronous orbit[9].

The work described here is unique in that it directly addresses the issue of exploiting the multi-track capability of certain types of sensors in order to maximize the number of track requests satisfied.

V. REFERENCES

[1] Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smit, B., Fisher, F.. Barret, T., Stebbins, G. and Tran, D. (2000). *ASPEN- Automating Space Mission Operations using Automated Planning and Scheduling.* Presented at SpaceOps 2000, Toulouse, France.

[2] Barbulescu, L. , Howe , A., and Whitley, D., "AFSCN Scheduling: How the Problem and Solution Have Evolved," *Mathematical and Computer Modeling*, 2006.

[3] Chien, S., and Gratch, J., "Adaptive Problem-Solving for Large-Scale Scheduling Problems: A Case Study," *Journal Of Artificial Intelligence Research*, 1996.

[4] Clement, B.J. and Johnston, M.D., "The Deep Space Network Scheduling Problems," *Proceeding of the Innovative Applications of Artificial Intelligence*, 2005.

[5] Frank, J., Jonsson, A., Morris, R. and Smith, D., "Planning and Scheduling for Fleets of Earth Observing Satellites," *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space*, 2002.

[6] Globus, A. Crawford, J., Lohn, J. and Prior, A. "Scheduling Earth Observing Satellites with Evolutionary Algorithms," *International Conference on Space Mission Challenges for Information Technology*, 2003.

[7] Gooley, T.D., "Automating the Satellite Range Scheduling Process," Master Thesis, Air Force Institute of Technology, 1993.

[8] Miller, James G., "A new sensor allocation algorithm for the Space Surveillance Network," *74th MORS Symposium*, 2006.

[9] Abbot, R.I., and Wallace, T.P. "Decision Support in Space Situational Awareness," *Lincoln Laboratory Journal*, Vol. 16, No. 2, 2007, pp. 297-335.