Intelligent Space Surveillance Network (SSN) Scheduling

Richard Stottler[1]
Stottler Henke Associates, Inc., San Mateo, CA, 94404

This paper presents a description of the difficulties for automatic optimized scheduling for the Space Surveillance Network (SSN) and then describes a design and algorithm to overcome those difficulties. These difficulties includes a large number of objects to track, which requires an even larger number of observations to schedule, different types of observation tasks, different regimes of space (near earth and deep space) and different types of requirements for the different types of objects (e.g., some have frequent visit requirements). The solution must therefore be similarly multifaceted with different algorithms appropriate for the different types of regimes while still being combined into a single whole.

Nomenclature

DS	=	Deep Space
EO	=	Electro Optic
CBR	=	Case-Based Reasoning
FOV	=	Field of View
FR	=	Frequent Revisits
GEO	=	Geosynchronous/Geosynchronous Earth Orbit
HEO	=	Highly Elliptical Orbit
LEO	=	Low Earth Orbit
LOS	=	Line of Sight
MEO	=	Medium Earth Orbit
RCS	=	Radar Cross Sections
SOI	=	Space Object Identification
SSN	=	Space Surveillance Network
SST	=	Space Surveillance Telescope
XML	=	eXtensible Markup Language

I. Problem Description

There are many challenges to managing and scheduling the Space Surveillance Network (SSN), including impaired communications and antiquated and unique computer hardware and software that requires local scheduling at the sensor for most of the SSN. This prevents the automated ability to provide real-time responses to requests during the 24-hour execution cycle and requires that a human at the sensor site manually respond to such requests. It also prevents (or at least hinders) globally optimized scheduling. Other challenges are that there are non-dedicated sensors, which are shared with other missions such as ballistic missile defense and non-taskable sensors (e.g., space fences), which gather track data without being explicitly requested to do so. There is a requirement to balance competing needs of different functional uses of the same sensors such as the need to gather complementary information on each object, including complementary orbit metric data, complementary Space Object Identification (SOI) data such as radar and optical images and signature data (such as Radar Cross Section (RCS) at different radar frequencies and different spectral intensities). The tasking is probabilistic in the sense that a sensor may or may not

[1] President, 951 Mariners Island Blvd., Suite 360, San Mateo, CA 94404, AIAA Member.

attempt to track the object requested and if it does attempt it, it may or may not be successful. The sensors are heterogeneous and complementary (radar: accurate range/range rate; EO: accurate angles and angle rate) and orbit accuracy requires different sensors and/or sensing different parts of the object's orbit.

The current scheduling process, using the current Tasker software, takes as input the current Space Catalog of about 20,000 objects. Based on the characteristics of each object, the specifics of each SSN sensor, and the geometry of each object's orbit with respect to each sensor site, a maximum probability of detection is calculated. Based on each object's priority, number of tracks and observations needed, and the maximum detection probabilities, the current Tasker tasks each sensor to track each object a specific number of times. A scheduler at each sensor site determines which opportunities and when within each opportunity to track (i.e., it schedules), without reference to which other sensors are tracking the same object (and when) and which orbit metrics will contain the most and least errors. It does not necessarily schedule the same time or even the same opportunity that corresponds with the maximum probability of detection.

There are several issues with the current process. The schedule is not globally optimized since scheduling occurs only locally without reference to what is assigned or scheduled at other sensor sites. A globally optimized schedule would typically have more tracks with the same resources. Furthermore, by considering the effects of other observations of the same object by other sensors at different parts of the orbit, specific sensors at specific times can be chosen that provide complementary observations of the object that will reduce the object's orbit metric error covariance matrix[1]. A simple example is that radar tends to be most precise in sensing the range and range rate whereas optical sensors are very accurate with respect to the angles and angular rates. Two nearby observations by these two different types of sensors will provide far more precise state information (and therefore smaller covariances) than two nearby radar observations would. The same effect applies to different parts of the orbit of a single object. The concept of complementary observations to minimize covariances is not currently utilized.

There are other problems with the current process. As noted earlier there is no ability to provide automated responses to sensor requests, whatever their priority, during the 24-hour execution cycle that also prevents an automated feedback loop solution. New sensors are being developed that can't be tasked by the old tasking software, including surveillance-based systems such as the S-band Fence and the Space Surveillance Telescope (SST). Frequent Revisit (FR) objects are handled with an all-passes code, which tasks every sensor every pass, which is duplicative and wasteful. Specifically there is little time separation between tracks when a Low Earth Orbit (LEO) object passes over the continental United States, providing little benefit. Worse, three sensors that are unique in their capability to track small debris objects waste time and energy on these Low Earth Orbit (LEO) FR objects. Finally, as new sensors are developed that provide the ability to track smaller and smaller objects, the number of objects in the space catalog will grow beyond the current system's ability to task for them.

For the purposes of this discussion we will term the requirements to track objects, "requests" where, specifically, a single request is the requirement to track one specific object once or to search a specific volume of space for a specific amount of time. Constraints on assigning resources to requests typically involve Line of Sight (LOS) (timing) issues involving orbital mechanics and also generally include a probability of detection calculations involving range, angles, cross section, and sensor sensitivity and precision. The Space Catalog is generally divided between Deep Space (DS) and LEO objects with the dividing line being a period of 225 minutes. It is generally assumed that one set of sensors is used for LEO objects and a separate set is used for DS objects although this is not completely true. Highly Elliptical Orbit (HEO) objects may qualify as DS objects but pass near enough to the Earth at perigee that LEO sensors can track them then. Due to their greater distance, DS objects require sensors of greater sensitivity and precision, require longer tracking time, and there are fewer of these sensors, hence the greater need for highly efficient and complementary scheduling. One especially import part of the DS region is the Geosynchronous (GEO) belt. (GEO can also refer to "Geosynchronous Earth Orbit.") Within each category (LEO, GEO, HEO, and DS) the timing constraints are similar. LEO objects orbit several times a day, have relatively narrow (10-15 minutes) time windows with LOS to specific locations on the ground, and are relatively near to those locations. The set of locations that a LEO object has LOS with is small and rapidly changing. GEO objects orbit once per day, meaning they are essentially fixed in longitude. This means that they have LOS to a large, relatively unchanging fraction of the earth all the time but at a long range. Non-GEO DS objects tend to have long time windows of LOS to specific locations and the set of those locations changes relatively slowly over time. Sensor requests typically inherit the constraints for the types of orbiting object they relate to. Resource management involves optimally tasking specific resources to meet the requests while obeying all of the constraints.

The schedule and scheduler must be flexible. Various high priority requests, including those related to in-space emergencies, may occur, which might require near-instantaneous changes to the schedule. Sensors may also break down, requiring real-time schedule changes. A very common occurrence is failure to track the requested object. In some cases this might require the scheduling of additional tracking attempts. So the schedule should have some built-in flexibility to handle emergencies and other real-time events. In addition, the scheduler must also be flexible by being able to reschedule based on these real-time events quickly. Being able to respond within the 24-hour execution cycle in order to be responsive implies that scheduling will be done in cycles. A new 24-hour schedule might be generated every few hours or it might be rescheduled immediately in response to high priority events. Either case however implies that during rescheduling, already scheduled tasks will exist and a decision be made to either change them or leave them as-is. The scheduler should also have the long-term flexibility to easily accommodate new resources and new types of resources as these are developed.

Scheduling problems involving meaningful resource selection are NP-Complete, meaning that the computational time of any algorithm guaranteed to produce an optimal solution is exponential with the number of those decisions. It is straightforward to build bad scheduling systems but difficult to build good ones. I.e., there are relatively simple algorithms, such as priority based schemes, which will generate correct schedules but not very good ones in the sense that many requests will be rejected that could have been scheduled with a better algorithm and that important metrics to optimize (SOIs achieved, average orbit metric covariance, average revisit interval, etc.) will be far from optimal. It is relatively difficult to build good scheduling systems because the NP-Completeness necessitates the use of good heuristics.

Many scheduling systems use a naïve approach to assigning resources to requests where the requests are fulfilled in priority order. If the current request cannot be scheduled, earlier-processed (and therefore higher priority) requests will not be reconsidered in order to try to accommodate the current request. In particular, if the current primary request cannot be scheduled, the request will be left unscheduled, even if it had been physically possible to shuffle some of the higher-priority scheduled requests to accommodate the current one.

II. Overview of a Solution

A new generation SSN scheduler must actually consist of a family of interrelated schedulers because different parts of the SSN scheduling problem have different performance measures and goals and different sensors are more schedulable than others. Therefore, different resource and time selection criteria are appropriate for different kinds of observation requests. Scheduling algorithms to optimize LEO frequent revisit sensing should minimize or eliminate the use of sensors better utilized on small debris sensing as well as minimizing redundant observations from other sensors. Scheduling for DS frequent revisit optimization must utilize predictions of surveillance search results and efficiently schedule space-based optical and ground-based radar and optical sensors by considering what parts of the GEO belt are in sunlight and which sensors are in solar exclusion. Scheduling of the remaining LEO objects should attempt to optimize the number of tracks per day for each object. Scheduling the DS objects should optimize some measure of the accuracy of the known position of each object such as the current, combined covariance matrix. This will involve seeking complementary measurements. Scheduling SOI requests should maximize total (complementary) information collected for each object. The value of each type of observation may depend on the attributes of the type of object being identified.

The use of the Aurora intelligent scheduling framework facilitates this family of interrelated schedulers because it allows for mixing and matching of different scheduling algorithms. In particular, different types of requests, can use totally different resource and time window selection criteria and methods. An Aurora-based intelligent SSN scheduling application would have one ordered queue of observation requests (for both orbit metric and SOI) to schedule and one global set of resources to schedule to meet those requests. A preprocessor would initially utilize predictions of which untasked sensors would likely track which objects (so that this information can be factored in to how many and which observations each object actually needs). The preprocessor considers both these predicted future observations and recent previous successful observations to determine a preferred (complementary) observation set for each object. Based on this, the preprocessor, using the bottleneck calculation described further below, calculates the resources and times of most contention and the requests that most need those resources at those times. At the same time, it also roughly calculates, required global resource utilization and the utilization for

different types of resources and tasks. This gives an initial prediction of the fraction of requests that are satisfiable. (E.g., if the required utilization of sensor resources is 120%, around 20% of the requests can't be met). This information is used to reduce the number of requests for the types, which will definitely not be met. This will be done differently based on the type of object (DS/LEO, FR). When reducing the number of requests for an object, its priority, recent set of successful observations (and current SOI and metric covariance states), and specific resources it could utilize are considered. Obviously, there is no reason to reduce requests for resources that are not over contended for. Other considerations are to prevent starvation and preserve requests for higher priority objects and those with currently poor (e.g., old) information.

As mentioned previously, many SSN sensors can only be tasked (i.e., told a desired number of observations and priority for each object) and not scheduled (i.e., told to observe a specific object at a specific time) and do make the specific scheduling decisions themselves locally. Some degree of scheduling control can be executed over these sensors, especially for the LEO case, which is the great majority of them. Through the following scheme, the global scheduler can either dictate a single observation opportunity to the sensor or a single specific set of adjacent opportunities (e.g., the 3 opportunities in a row constituting the 4^{th}, 5^{th}, and 6^{th} opportunities of the day. The scheme is relatively simple. The sensor's 24-hour day is divided up into time periods corresponding to the shortest LEO orbital period, about 90 minutes. The first time period is designated for the objects sent to the sensor with top priority (which would not necessarily correspond to their actual priorities as input by the users). The second time period would be for all top priority objects requiring 2 observations and second priority objects. The third time period would be for all top priority objects requiring 3 observations, all second priority objects requiring 2 observations, and all third priority objects. The fourth and subsequent time periods and priorities are defined similarly. To schedule an object, for example, at the fifth time period, simple give it a priority of 5 and a number of observations of 1. Alternatively, to schedule that object in each of the consecutive opportunities, starting in the fifth time period, give it a number of observations of 3. For this scheme to work, the global scheduler does need to fill each time period with observations.

There are two main categories of conflicts: those that can be resolved while satisfying the requests, presumably by shuffling other resource assignments in allowable ways and those that simply cannot. Resolving the former is relatively straightforward but does involve the use of good heuristic algorithms to produce as good a result as possible. Resolving the latter requires actually deciding what requests should have their constraints relaxed somewhat. These resource contention issues essentially constitute potential bottlenecks where specific resources could likely experience a situation in which for specific periods of time they are being requested for more service than they can provide. The scheduler should try to make resource and time window selections that avoid these bottlenecks.

Since, like most NP complete problems, generating a schedule is much harder and more time-consuming than testing or grading one, it would be possible for the scheduler to include several different scheduling algorithms, all of which were independently generating schedules. Relatively simple and quickly executing grading software could then evaluate each schedule and select the best one. One algorithm might be the bottleneck avoidance algorithm (including swapping out earlier processed, lower priority tasks), described later. In situations where it is possible to service all tasks with proper consideration of resource contention, the best solution might be from the bottleneck avoidance algorithms. Nevertheless, there may also be some situations when all requests cannot be met in which another algorithm might perform better.

III. Objectives

The ultimate goal would be to improve the performance of the SSN, including improved space catalog accuracy and precision and improved SOI information through more and more complementary observations, better responsiveness to real-time observation requests and situation changes, and better tracking of small debris in LEO through more efficient use of the sensors that are able to perform this function. Other improvements would include more efficient tracking of DS frequent revisit objects, handling increased number of objects and new types of sensors, and taking full advantage of improved communication and control when it is available to globally optimize the SSN sensor schedule.

Since many of these techniques and algorithms have already been proven in a prototype[1], the next logical step is to develop the full-scale family of SSN schedulers and integrate them with the appropriate SSN systems. These

include the LEO frequent revisit optimizing scheduler, the DS frequent revisit optimizing scheduler, and the SSN Global Optimizing Scheduler. The intent is to eventually have these capabilities deployed in an operational system in order to provide improved SSN scheduling, integrate each into the appropriate environment, show superior performance on SSN scheduling problems, and demonstrate significant benefit for SSN reactivity and space catalog quality.

IV. System Description

The High Level Architecture for each of the specific schedulers is shown in Fig. 1 below. The modules can be assembled in an existing intelligent scheduling system architecture, which provides for customization of each decision point in the scheduling processing. For example, it handles resource usage profiles and visibility requirements, provides for pluggable resource/time window selection methods and satisfaction of temporal, spatial, and arbitrary constraints, and includes the notion of scheduling cycles, an evolving schedule, and the need for a separate preprocessing module.

An SSN application desiring scheduling provides an eXtensible Markup Language (XML) (or other agreed on format) description of the scheduling problem including a description of the tasks to be scheduled, the resources available and the constraints. One type of common constraint for SSN applications is visibility. Normally these are calculated and stored in a separate file that is merely referenced in the XML description of the scheduling problem. However, mere transmittal of the task to the scheduler does not ensure that the required resources can be found to execute it. Whether enough resources exist to schedule all requested tasks is based on the number of resources available (and the optimality of the scheduling algorithm as shown in the earlier simple scheduling example). Associated with each task are its resource requirements and temporal, spatial, and other constraints. A typical temporal constraint, for example, is that one particular task must be complete before another can begin (such as sensor configuration, mode selection, slewing, and other setup occurring before the sensor event can execute). Another example is that a specific task must be completed by a certain time (e.g., a specific tracking task must be completed by midnight). Constraints may exist associated with tasks, resources or both. For example, a constraint may state that a particular task must use a specific sensor at a specific location.

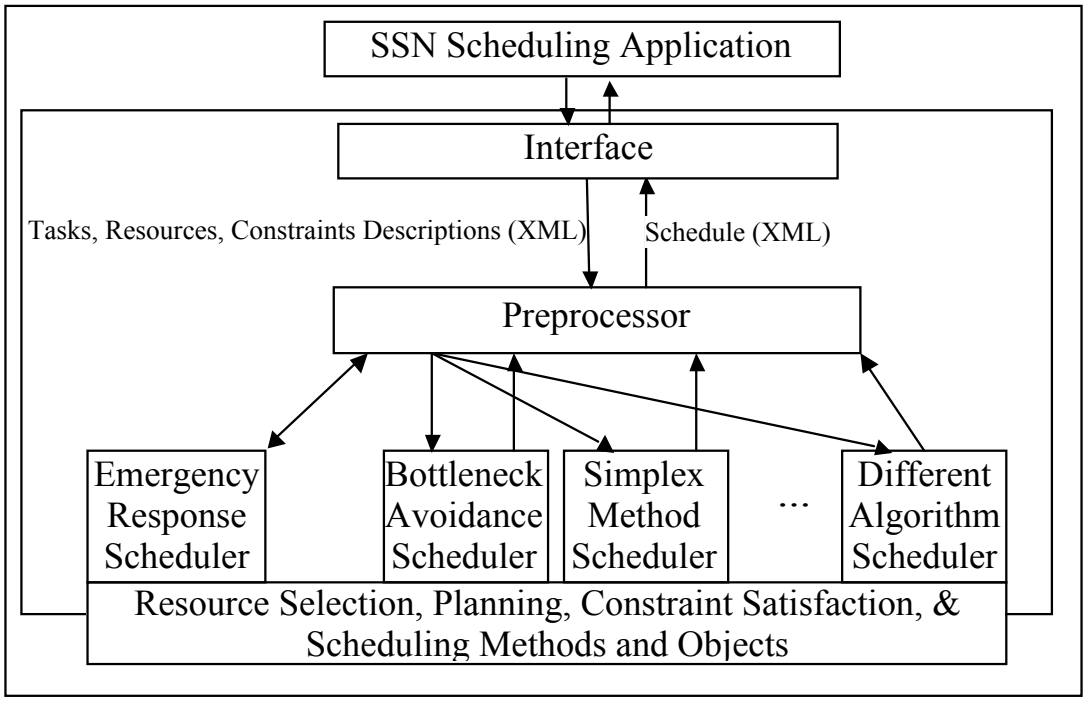

Figure 1. High Level Scheduler Architecture.

The roles of the preprocessor in addition to those described above are to call each applicable scheduler, grade each returned schedule, and select the best schedule. Emergency requests requiring attention immediately would be passed to the Emergency Response Scheduler for immediate scheduling, so that the appropriate tasks can be passed back and executed immediately and the rest of the near-term, already-published schedule minimally disturbed. Otherwise, or for the remaining requests, the Preprocessor selects one or more appropriate Sub schedulers. These Subschedulers will all be independent and therefore would easily fit into a distributed architecture with each Subscheduler having its own computing resources, if required.

The grading scheme for comparing two schedules can be more or less complex and does not have to be uniform across the space catalog objects. We anticipate that our metrics for measuring the quality of produced schedules will be different for different areas of space and different types of requests. For DS object metric requests, some measure of the accuracy of the known position of the object such as the current, combined covariance matrix would be used. Formulas or software for this calculation provided by the existing SSN scheduling and/or covariance calculation community could be used. For LEO objects, simply the number of tracks per day for each object could be used. There are empirical tables that link this simple metric to historically measured errors in tracked objects. For FR objects, the maximum and average time between observations would be used. SOI requests would have their own metrics appropriate for them.

Each Subscheduler conceptually takes as input the tasks, resources and constraints and produces, as output, the assignments to the tasks of resources and time windows that meet the constraints. Each task description includes what resources it requires and for how long and the constraints that it is involved in. In practice, most of the resources needed by tasks will have been predefined with only updates as to their availability being passed in (e.g., radar failure) at scheduling time. The Scheduler generally maintains its own record of the availability of each resource based on the tasks and the time windows currently assigned or tentatively assigned to them. The constraints may be hard (an absolute requirement) or soft (satisfaction is desired but not absolutely necessary) and may be temporal, resource-oriented, or of miscellaneous type.

Each Subscheduler has several decision points, each of which our generic intelligent scheduling architecture allows to be customized through selection of existing options or plugging in new software. One major set of decisions is what the scheduler should process and in what order to process. One common solution is to process tasks in due date or priority order. Another possibility is to process resources in the order in which they become available. Still another is to process constraints, the tightest ones first. We expect as described in the algorithm description below to process tracking requests, sorted so that the tasks that request the most constrained resources at the most constrained times are scheduled first. It is extremely important to note that the order in which processing will occur has a very large effect on the quality of the solution and that simple priority or time ordering schemes will often produce very poor results in any reasonably complex scheduling situation. Additional heuristics are also helpful to include in the ordering decision.

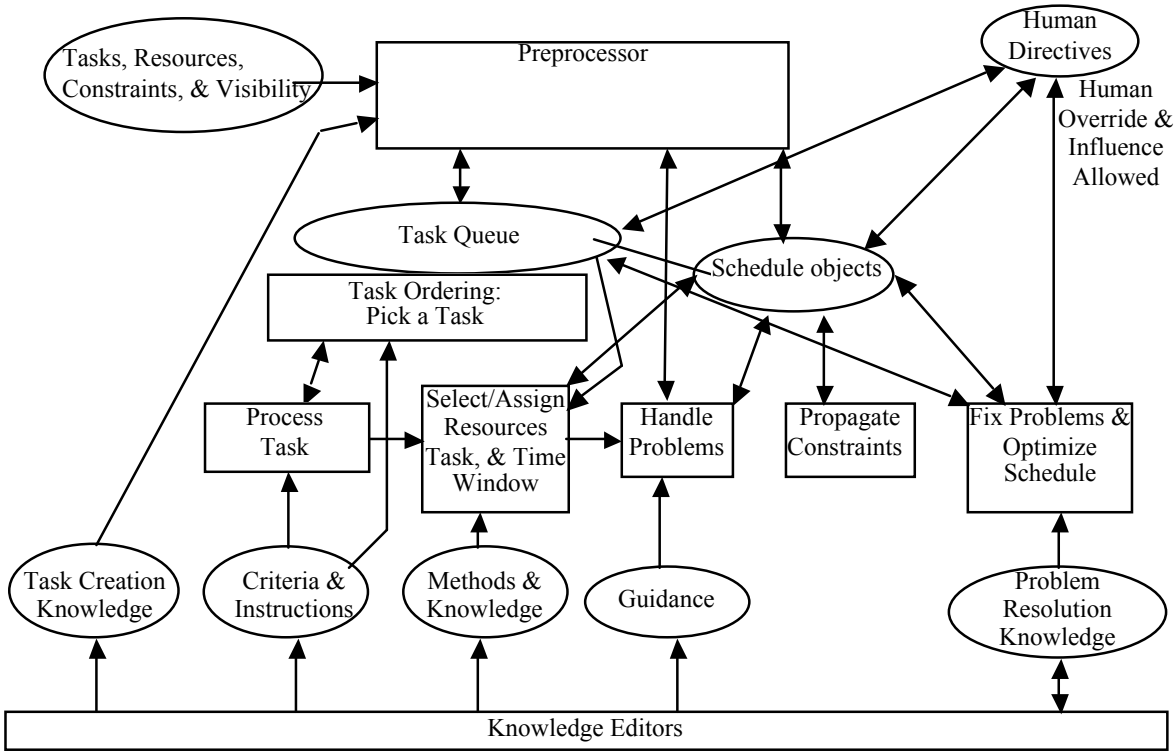

Figure 2. Internal Scheduler Architecture.

Another major decision point is assigning to a task the needed resources and time windows, while satisfying the applicable constraints. Usually the time window is constrained by the need to get a task done by a certain time while having to wait for events or other tasks to finish before they can start. There may be an additional soft constraint to have the task finish as soon as possible or to start as late as possible. Usually there is at least some freedom in choosing the time window at the same time that the specific needed resources are selected. Choosing the set of resources that are available at the same time that best meet the constraints and that are best for the overall plan can be difficult. "Greedy" systems that choose what is best for the specific task or resource often produce poor results, overall. Some systems use a resource balancing approach whereby the least utilized resource is chosen, under the assumption that if a later-processed task needs a specific resource instance, all of them will still be available; but this can also run into problems, depending on the processing order, especially in cases like the SSN where resources are always tight. A better approach is to choose the resource that other tasks are least likely to need and at the time at which they are least likely to need them. This is the approach in the algorithm described below.

The processing queue is reordered dynamically during the scheduling process and is largely based on contention modified by several factors. Contention is calculated by first "probabilistically" allocating each request across the possible sensors and time windows (based on LOS calculations), weighted by the quality, complementariness, and success probability of the observation and then for each sensor, summing up all possible "probabilistic" allocations. As specific scheduling decisions are made these bottleneck calculations have to be updated for the object just scheduled. These updates have to take into account that 1) the specific request has now been 100% scheduled at a specific sensor and a specific time (and therefore is not "partially" scheduled anywhere else) and 2) the complementariness of other possible observations are changed to reflect the data that will likely be gathered by this observation. The bottleneck calculation is not just used for the order of processing but is also used to select specific resources and times of observations to avoid the bottleneck regions (while still balancing the need for quality, complementary, probable observations).

Most SSN sensors can either track a very large number of objects within a wide, fixed Field of View (FOV) or

track one object at a time. But some SSNs sensors (such as the SST) can opportunistically track more than one object with a movable FOV. This capability can be leveraged in one of two ways, depending on how important a factor it is. The simplest way to leverage it is simply opportunistically. Whenever a tracking task is scheduled on such a sensor, nearby objects are checked to see if the FOV can be changed to include the target object and as many others as possible. Any other ones that are captured don't need to be scheduled in the near future and/or may be able to be unscheduled (if they were already scheduled) from a nearby time.

If this phenomenon is expected to have a large effect, it may make sense to implement a more complicated algorithm whereby this issue is used more centrally in the scheduling process[2]. Briefly 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 it can obtain as many of the tracks with which they may be potentially 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 in order to satisfy as many track requests as possible, while conforming to the requirements concerning priority, number of tracks and number of observations per track. The approach from reference 2 to this scheduling task 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. The approach defines techniques for clustering objects whose visibility sets intersect. These intersections are discrete subsets of the search space that 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.

One major decision point arises when, during the assignment of resources and time window to a task, no reasonable assignment can be found. This may involve unsatisfiability of a soft constraint in which case it will most likely be ignored but may also involve the inability to satisfy hard constraints or to find needed resources at the required time. In one of these latter cases, the appropriate action for the planner depends on the aspects of the domain, the way it is modeled, the scheduling time available, and the optimality of the other algorithms in use. For example, if an unavailable resource is not completely saturated, it may be appropriate to try to shuffle some of the other task assignments to make a "hole" in a resource availability profile for use by this task. Perhaps a lower priority or less important task that is already scheduled should be unscheduled to make room for this task. The concept of changing the plan in the same cycle brings up the notion of backtracking. A small amount of backtracking can be important to optimize a plan in relatively small ways, but overuse of this technique degenerates into systematic search, which should be avoided for most complex problems. Often heuristics can be efficiently used to handle the problem of resource/time window unavailability. These take two forms. One is that often some of the supposed hard constraints or resource requirements have some leeway. For example, a sensor setup task may be specified to take 60 seconds but in an extreme situation, 30 seconds might be acceptable if the only alternative is not to have the sensor event occur at all. The other form of heuristic is one whereby all resource requests and constraints can be met by a clever trick that the human planner has found to be useful in similar situations; e.g., human planners have found that GEO and Medium Earth Orbit (MEO) related requests are the most likely to be able to be moved to make room for another, more tightly constrained LEO related request. (Of course, our proposed resource assignment algorithms already take this into account the first time through.) These latter forms of heuristics are less likely to be useful if the ordering and selection algorithms are near optimal. Ultimately it may be necessary to place the task in question (or another one swapped out to make room) on an unschedulable list to be dealt with at the end of the cycle.

After each item has been processed, another decision point is reached: that of optimizing the plan just produced before the resource assignment cycle ends. This includes trying to solve the remaining problems and optimizing the soft constraints. Recall that it is possible that acceptable resources and time windows could not be found for some tasks. Thus, tasks may be on an Unscheduled List or may be scheduled in such a way as to create conflicts and otherwise violate constraints. A Case-Based Reasoning (CBR) module exists to suggest conflict resolution strategies based on previous similar conflicts. These may be able to be applied by the scheduler itself to fix the conflict or may require user approval. Alternatively policy rules may dictate how to resolve conflicts, such as swapping out lower priority requests to allow higher priority ones to be satisfied. Currently, human schedulers manually manage additional high priority lists to ensure that specific objects are adequately tasked. These human schedulers may

continue to be involved in the scheduling process and the reuse of their past directives for solving past conflicts may be useful for suggesting solutions to current similar conflicts.

V. Summary

Space Surveillance Network sensor scheduling is complex and voluminous with the object and request space divided into separate portions with each having its own objectives, metrics, and parameters such that different scheduling algorithms are appropriate for the different portions. However, these must be combined into one integrated whole since aspects of each portion impact the others. A method and design for accomplishing this and descriptions of the algorithms were presented.

References

[1]Stottler, R., Thompson, R., "Globally Optimized Scheduling for Space Object Tracking," *Proceedings of the Infotech@Aerospace 2011 Conference,* Vol. 1, AIAA, Reston, VA, 2011.

[2]Mohammed, J., Stottler, R., "Rapid Scheduling of Multi-tracking Sensors for a Responsive Satellite Surveillance Network," *Proceedings of the Infotech@Aerospace 2010 Conference*, Vol. 1, AIAA, Reston, VA, 2010.