Automatic, Rapid Replanning of Satellite Operations for Space Situational Awareness (SSA)

Richard Stottler
Stottler Henke Associates, Inc., San Mateo, CA 94002
Kyle Mahan
Stottler Henke Associates, Inc., San Mateo, CA 94002

ABSTRACT

An important component of Space Situational Awareness (SSA) is knowledge of the status and tasking of blue forces (e.g. satellites and ground stations) and the rapid determination of the impacts of real or hypothetical changes and the ability to quickly replan based on those changes. For example, if an antenna goes down (either for benign reasons or from purposeful interference) determining which missions will be impacted is important. It is not simply the set of missions that were scheduled to utilize that antenna, because highly expert human schedulers will respond to the outage by intelligently replanning the real-time schedule. Similarly if there is a vehicle emergency or a vehicle must be quickly maneuvered because of a possible collision, in addition to scheduling the required supports, minimizing the impact of the added support must also be performed We have developed an automatic scheduling and deconfliction engine, called MIDAS (for Managed Intelligent Deconfliction And Scheduling) that interfaces to the current legacy system (ESD 2.7) which can perform this replanning function automatically. In addition to determining the impact of failed resources, MIDAS can also replan in response to a satellite under attack. In this situation, additional supports must be quickly scheduled and executed (while minimizing impacts to other missions). Because MIDAS is a fully automatic system, replacing a current human labor-intensive process, and provides very rapid turnaround (seconds) it can also be used by commanders to consider what-if questions and focus limited protection resources on the most critical resources. For example, the commander can determine the impact of a successful attack on one of two ground stations and place heavier emphasis on protecting the station whose loss would create the most severe impacts. The system is currently transitioning to operational use. The MIDAS system and its interface to the legacy ESD 2.7 system will be described along with the ConOps for different types of detailed operational scenarios.

1. INTRODUCTION

Building a conflict-free schedule from a large set of satellite communication requests is a difficult problem. Human schedulers are very adept at generating high-quality solutions, usually allowing all requests to be serviced. However this process is time-intensive and requires highly trained, experienced individuals, and the demands placed on them will only increase as demand intensifies. There is an opportunity for an automated scheduling tool to take some of the burden off these schedulers. Additionally such a tool could provide the currently-unavailable possibility of running "what-if" scenarios to assess the impact of a potential event or mission change.

In this paper, we discuss a multiple-pass scheduling algorithm which consists of two steps. The first is "Bottleneck Scheduling" that minimizes inter-support conflicts while obeying all user-specified constraints. This eliminates many conflicts and reduces overall resource contention. However, this is only the beginning in solving the problem of generating a near-optimal solution. Schedulers frequently (tens to hundreds of times a day) encounter situations where conflicts cannot be solved without modifying the constraints of the original request – they must "bend the rules". These modifications must be reviewed and accepted by the user (satellite operations center) before they can be included in the final schedule, so it is important that the scheduler have a high degree of certainty that his or her suggestions will be accepted.

In this paper, we also discuss the second step, based on the concept of business rules – archetypal strategies that describe the different ways in which constraints can be modified –, and how they can be orchestrated to solve even complex collisions between requests. There are a limited set of business rule types (on the order of 10). For example, a support may be given less setup time than the user has requested. It can be moved temporally, to another side, or to another station altogether. Long supports can (and often must) be broken up and handed off from one

station to another. Each of these business rules takes a variety of parameters that describe when and how much the strategy can be applied based on the type of support, the user, and who they are in conflict with.

Many conflicts can be solved by simply iterating through all conflicted supports and applying business rules in order until a solution is found. However, applying the rules in combination (e.g., combining a less severe rule with a more severe rule, rather than applying the severe rule alone) will often yield more acceptable results. And in some cases a solution can only be reached by considering cascading combinations of business rules. Fortunately only certain rules make sense in combination, so it is possible to consider the impact and application of each pair.

2. BOTTLENECK AVOIDANCE INTRODUCTION

Bottleneck Avoidance (BA) is an elegant and relatively intuitive approach to scheduling that is intended to mimic the decision processes of human expert schedulers. It produces a high quality schedule quickly without backtracking. The technique was originally inspired by [1] and has since been developed as an approach to a variety of scheduling domains where there is a high degree of resource contention. BA utilizes a specialized data structure for tracking not only the current resource allocation of scheduled tasks, but also the predicted or probabilistic allocations of unscheduled tasks. This provides the algorithm with a broad view of the schedule space, taking into account *all* tasks, rather than limiting its analysis to tasks that have been previously scheduled.

BA informs several different decision points in the scheduling process: processing order (the order in which tasks are considered for scheduling), temporal assignment, and resource selection. In all of these stages, the target heuristic is the reduction of resource contention. Contention is the degree to which multiple tasks are likely to be assigned to the same resource simultaneously, and areas of particularly high contention are called "bottlenecks". BA works by first identifying the worst bottlenecks, identifying the tasks that contribute most to these bottlenecks, and scheduling them in a way that reduces the bottlenecks if possible. By repeating for each task, a high quality schedule can be built in a single pass without costly backtracking or search.

3. PROBABILISTIC BOTTLENCK MODELLING

BA utilizes a specialized data structure for tracking both the *actual* and the *predicted* allocation for each resource. Each resource is modeled using a sorted tree of "time slots", contiguous blocks of time that know their actual and predicted/probable allocation, as well as a list of their actual and potential users (tasks).

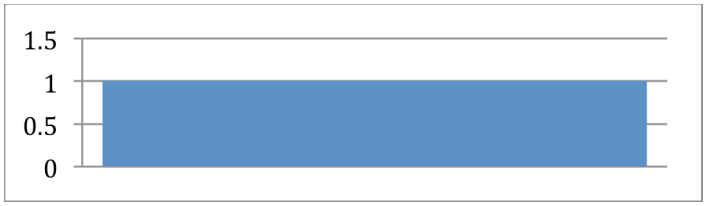

Fig. 1. Predicted allocation of an inflexible task. *A task with one possible temporal allocation is represented by a block of predicted usage with quantity 1.*

In the preprocessing phase of scheduling, Bottleneck Avoidance looks at each task and determines **a)** its earliest possible start date and latest possible end date (based on explicit constraints and its satellite's visibility, if applicable), and **b)** all of its possible resource assignments. The task is then added as a potential user to every time slot in its possible time window on every resource where it might be allocated. A task with a fixed time window and only one possible resource assignment is added to that resource with a predicted quantity of 1, meaning it will absolutely be scheduled in that position (see Fig. **1**). On the other hand, tasks with several alternative resource assignments will have their contribution to the resources' predicted usages reduced

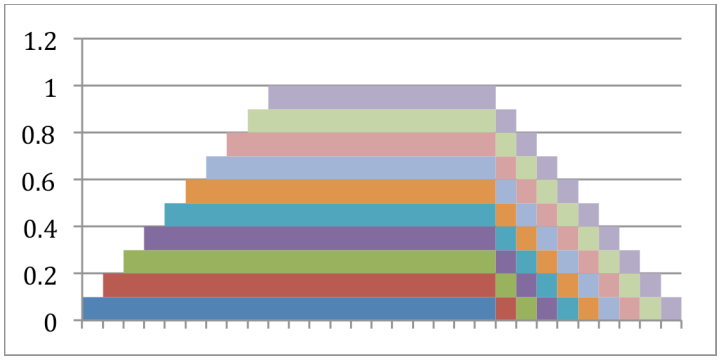

Fig. 2. Calculating the predicted usage of a 20-minute task in a 30-minute window. *Allocate a 10% probability to each possible start time and sum them. Because every possible allocation includes the middle 10 minutes, the trapezoid plateaus at 1.0 — 100% probability.*

accordingly. So if for example, the task in Fig. **1** were allowed to schedule on either one of two sides, it would contribute only 0.5 to each side's predicted quantity, meaning that there is a 50% probability that the task will be scheduled in that position on each side.

It becomes somewhat more complicated when dealing with flexible temporal assignments. Take for example a 20-minute task with an early start of 0000 and a late end of 0030. The task can start at any minute between 0000 and 0009, inclusive, to finish before the late end time. Examining only whole minutes (no seconds), this accounts for 10 possible assignments. If we assume that no start time is more likely than any other, then each has a 10% probability. By summing the usage profiles for these 10 possible assignments, we obtain an overall profile approximating a trapezoid, as in Fig. **2**. This means that there is a 10% probability of the task being assigned at time 0000 but a 100% probability of the task being assigned at time 0015 (because all possible allocations must include 0015). It is worth noting that the area of this trapezoid is:

$$^1/_2\,(10\ minutes \times 100\%) + (10\ minutes \times 100\%) + \ ^1/_2\,(10\ minutes \times 100\%) = 20\ minutes \times 100\%$$

or 100% of the duration of the task, which is the same result as if the task were a fixed duration. We can generalize this and say that the area under the predicted usage curve (for a single resource) should always add up to 100% times the duration of the task, if the task can only use that single resource. For tasks that can use one of several resources, the total area under the predicted usage curves for all of those resources should add up to 100% times the duration of the task.

For a task with a possible window significantly larger than its duration, this implies that its maximum usage will be somewhat less than 100%. As an example of this, take a 10-minute task in the same 30-minute window. This task can start at any time from 0000 to 0019 – 20 possible assignments – each with a probability of being selected of 5%. Summing these profiles, we get the trapezoid in Fig. **3**, which plateaus at 50%. Any given minute from 0010 to 0019 only has a 50% probability of being assigned. As expected, the area under this curve is 100% of 10-minutes – the task's duration.

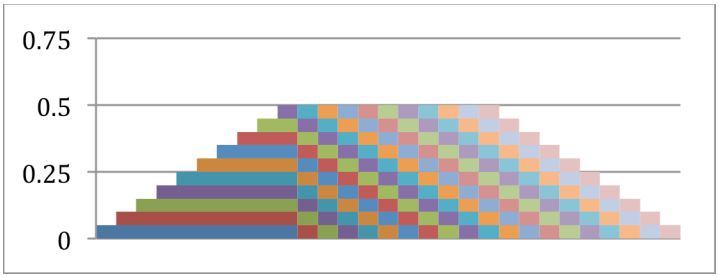

Fig. 3. Calculating the predicted usage of a 10-minute task in a 30-minute window. *Allocate 5% to each of 20 possible assignments. In this case, the trapezoid plateaus at 0.5 – there is a 50% probability that any minute in the middle section will be allocated.*

It turns out that the critical difference between these two examples is that in the latter, the task's duration is less than half of the possible window's size. Using this knowledge, calculating the predicted curve for any task is straightforward; refer to Fig. **4** and the following equations:

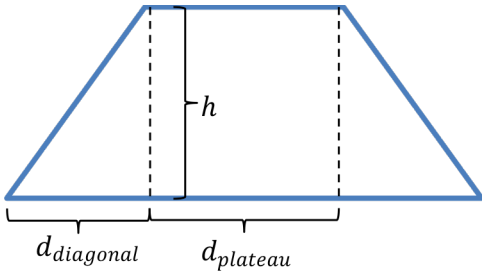

Fig. 4. Trapezoid with labels.

$$t_{start} = possible\ window\ start$$
$$t_{end} = possible\ window\ end$$
$$d_{task} = task\ duration$$
$$d_{window} = t_{end} - t_{start}$$
$$d_{diagonal} = \begin{cases} ^1/_2\,(d_{window} - d_{task}), & d_{window} > 2d_{task} \\ (d_{window} - d_{task}), & d_{window} \leq 2d_{task} \end{cases}$$
$$d_{plateau} = d_{window} - 2d_{diagonal}$$
$$h = \frac{d_{task}}{d_{diagonal} + d_{plateau}}$$

Accounting for diagonals, rather than approximating predicted usage as a simple rectangle, has several advantages. When attempting to alleviate a bottleneck, it has the effect of pushing tasks to the outside of other tasks' possible windows, leaving the more flexible middle section open. Possibly more important for the satellite communication domain, it obviates a problem that comes up when dealing with sequential tasks: in this case the prepass that occurs immediately before a contact (a task that prepares the remote tracking station for communication with a satellite). Take two tasks that require the same resource, where the second task is constrained to start immediately after the first task ends. In Fig. **5**, we see on the left that the rectangle approximation erroneously predicts a bottleneck in the

overlap; whereas on the left, the trapezoid method correctly reflects the fact that it is not possible for the two tasks to schedule simultaneously. The task and prepass are effectively treated as if they were one continuous task by the predicted allocation model.

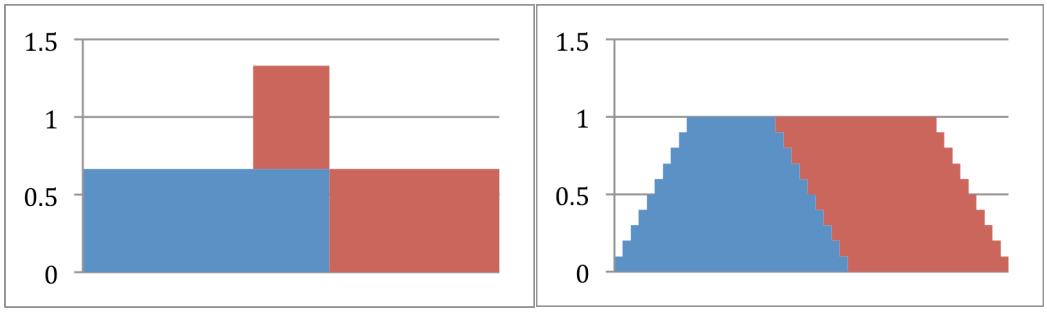

Fig. 5. The predicted usage of two sequential tasks, approximated with rectangles (left) and fully calculated using trapezoids (right). *The first diagram incorrectly predicts a conflict between the two tasks, while the second reflects the fact that it is not possible for the two tasks to schedule simultaneously. Note that unlike the previous figures, the two colors represent separate tasks rather than different possible allocations of a single task.*

4. SCHEDULE PROCESSING ORDER

The chronological order in which tasks are considered for scheduling is the Schedule Processing Order. Attaining a good processing order for tasks is critical to building a near-optimal schedule with a one-pass algorithm. A human expert likely uses some heuristic when deciding the order in which to review tasks. Some schedulers, for example, tend to look at Low-Earth-Orbit (LEO) contacts before High-Earth Orbit (HEO) or Geosynchronous (GEO) contacts. The reasoning behind this decision is that LEO satellites have comparatively short visibility windows, significantly limiting the temporal flexibility of their contacts. By getting them out of the way early, a scheduler ensures that the contact gets the resources it needs while leaving the more flexible tasks for later, when the resources have already partially allocated. If we waited until the end of scheduling to allocate these inflexible tasks, we would be left with very few (or no) alternatives if the required location is taken. Another way to say this is that, by scheduling inflexible tasks first, we keep the maximum amount of flexibility in the schedule at each step.

Bottleneck Avoidance uses a similar heuristic, attempting to schedule the least flexible tasks before the most flexible tasks. However, the bottleneck tracking data structure allows us to define "flexibility" using several dimensions: temporal flexibility (like the LEO-before-HEO approach), the degree of contention for resources in that time window, and the current state of tasks that have already been scheduled. These three considerations are automatically entailed in the predicted usage calculations for finding bottlenecks as shown in the following example. Fig. **6** contains a simplified scheduling problem with only one RTS with two sides. In this simplified example, the two sides are completely independent, such that a task on Side 1 cannot conflict with a task on Side 2 (real world scheduling requires that we consider equipment shared between the two sides, as will be discussed later). The diagram contains predicted usages for three separate tasks. The blue task is essentially "fixed" – it has no temporal nor resource flexibility at all. This might represent a LEO task that requires a specific side. The green task is a shorter task with a flexible time window; because the task can be scheduled on either side, its predicted contribution to each side is reduced by half. Because the task's duration is less than half of the possible time window's duration, its contribution is decreased even further (as in the example in Fig. **3** above). Finally the red task is a longer contact in a somewhat flexible time window. Unlike the green task, it has duration longer than half of its possible time window and therefore contributes 50% to either side (as in Fig. **2**).

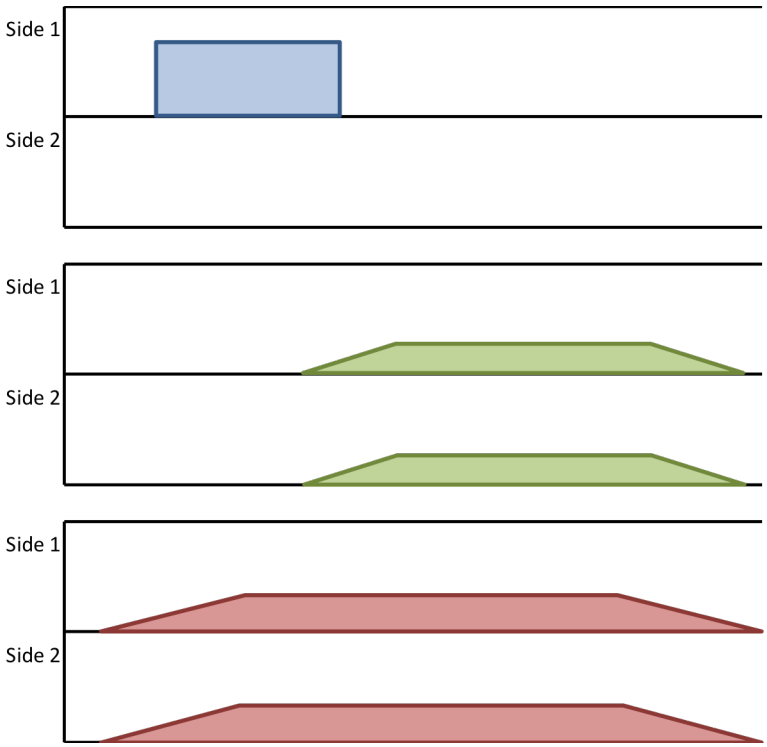

Fig. 6. Predicted allocations for three tasks in a simplified scheduling problem with only two sides. *The blue box represents a temporally inflexible task. The green trapezoid represents a short, flexible task, and the red trapezoid represents a long, flexible task. The blue task can only schedule on Side 1 while the red and green tasks can schedule on either side.*

The next step is to calculate the total predicted usage by adding the three profiles together. This aggregated usage is shown in Fig. **7** (with the three tasks' contributions still in their various colors). The dotted line represents a predicted usage of 1.0. This is not a magic number – a predicted usage greater than 1 does not guarantee a conflict, nor does a predicted usage less than 1 guarantee there will not be a conflict. But it does mean that of all possible assignments of the three tasks, a majority have a conflict in that region, i.e., we would predict a conflict there *on average*. Careful processing order and resource selection will avoid this.

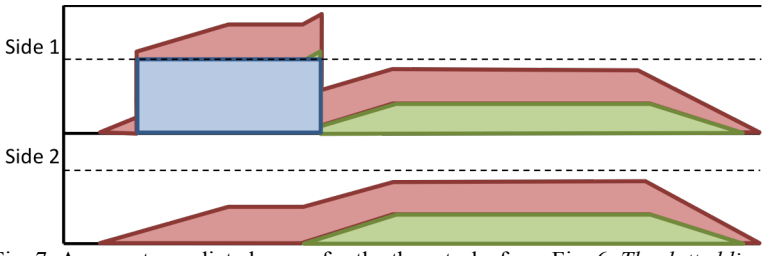

Fig. 7. Aggregate predicted usage for the three tasks from Fig. 6. *The dotted line represents a predicted usage of 1.*

Once the aggregate model is completely populated, the next step is to identify the largest bottlenecks. This is simply a matter of traversing the data structure[1] and searching for the highest peaks – these peaks represent the areas with the greatest resource contention. We are only concerned with peaks that include at least one unscheduled task (which all tasks are at this point), and typically a plateau is considered a larger bottleneck than an apex (a profile that rises to a point and then drops off) of equal height.

[1] We found a significant performance improvement by caching of the highest peak(s) for each resource and using this cached value on subsequent searches. The cache is cleared for a resource whenever a task is added or removed and recalculated the next time it is needed.

After selecting the highest peak or peaks, we select the unscheduled task that contributes the most to the peak (or to one of the peaks). Ties can be broken arbitrarily at this point, or some secondary criteria can be employed (e.g., a user specified priority). The greatest contributor is the least flexible task within the peak. If we are able to move it away from the peak, it will result in the greatest reduction in the overall peak size, and if we are not, it's better to determine that as early in the process as possible. Fig. **8** illustrates this procedure. The blue task is the largest contributor to the tallest peak and so is selected for scheduling first. Because it has only one possible assignment, selecting its scheduling location is trivial.

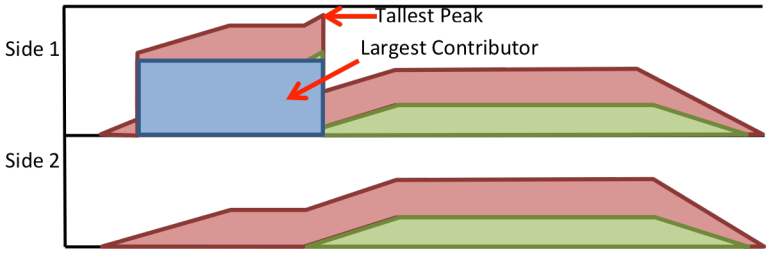

Fig. 8. Selecting the first task to schedule. *BA looks for the largest unscheduled contributor to the tallest peak.*

The blue task's predicted usage is removed from the bottleneck data structure and replaced with *actual* usage. Actual usage does not require any complicated trapezoid calculations; it is simply a single block in the task's selected time window on its selected resources. Intuitively it might make sense at this point to go through and update the green and red task's predicted profiles to account for the fact that they can no longer schedule where the blue task scheduled (in fact, the red task cannot schedule on Side 1 at all). The problem however is that this would require recalculating the usage profile for all tasks that overlap with the previously scheduled task, which in many cases would prove computationally infeasible. For this reason we will proceed with the predicted usage profiles as they were originally calculated, which works very well in most situations.

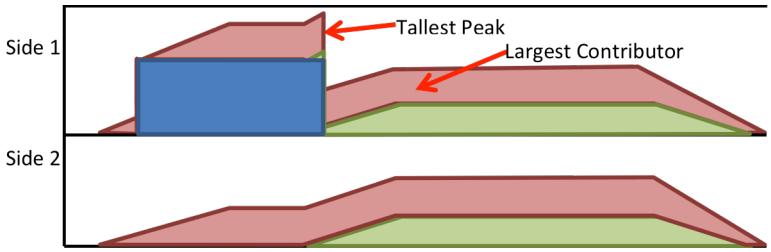

Fig. 9. Selecting the next task to schedule. *The highest peak is the same as before, but now the red task is now the largest unscheduled contributor.*

Fig. **9** shows the second round of processing order consideration. The blue task is now a darker color to indicate that it is an actual rather than predicted allocation in the bottleneck model. The highest peak remains the same, but the red task is now the largest unscheduled contributor, and as such it is selected as the next task to schedule. Because the red task has some flexibility, the choice of where to schedule it and which resources to select is more interesting.

5. TEMPORAL ALLOCATION AND RESOURCE SELECTION

Scheduling a task that has some flexibility requires allocating it on two dimensions:

1. In an appropriate temporal assignment, within its possible window
2. On a combination of resources that meet its requested requirements

The goal of this stage of Bottleneck Avoidance is to schedule the task such that it minimizes all possible bottlenecks. BA approaches this in two separate steps. First, we attempt to find all "open time spans" – time windows that are at least as long as the task's duration, during which a continuously available set of resources satisfies the task's requirements. A full discussion of this search mechanism is beyond the scope of this paper, but it involves "walking" all of the task's possible resources from the task's early start to late end, keeping track of the currently active time spans and the resources that are available continuously during each span. When a resource becomes unavailable, all spans that involve that resource are evaluated. If the span is at least as long as the task in question, it

is added to the list of open time spans. If, when the search is complete, no time span meeting these criteria has been found, it means that the task will schedule in conflict on at least one resource. In this case, we want to ensure that the task at least schedules during a valid visibility window, so we begin the search again, ignoring all resource requirements *and considering only satellite visibility*. In our current example, the red task can schedule anywhere within its time window on Side 2, so the search returns one and only one open time span.

Now we know that a valid resource assignment is possible on any time window within one of these open time spans (except in the case where we resorted to checking visibility windows only). The next step is to consider the bottleneck model to find an assignment that minimizes all bottleneck peaks as much as possible. For simplicity, the current task's predicted allocation is removed from the bottleneck model. Then we iteratively evaluate each possible time window within each open time span, calculating the tallest bottleneck peak in that window, as well as the total bottleneck area in the window (the integral of the bottleneck curve from the time window's start to its end). The window with the smallest peak is selected. If more than one window is found with the same lowest peak, bottleneck area is used as a tie break. If two candidate windows have the same peak and area, additional criteria can be used (e.g., distance from a user-specified preferred start time). Fig. **10** illustrates the open time span as well as a few candidate time windows within the span. All candidate time windows in the example will result in the same peak height, so area is used to break the tie. In this case, the first candidate window results in the smallest area and will be selected.

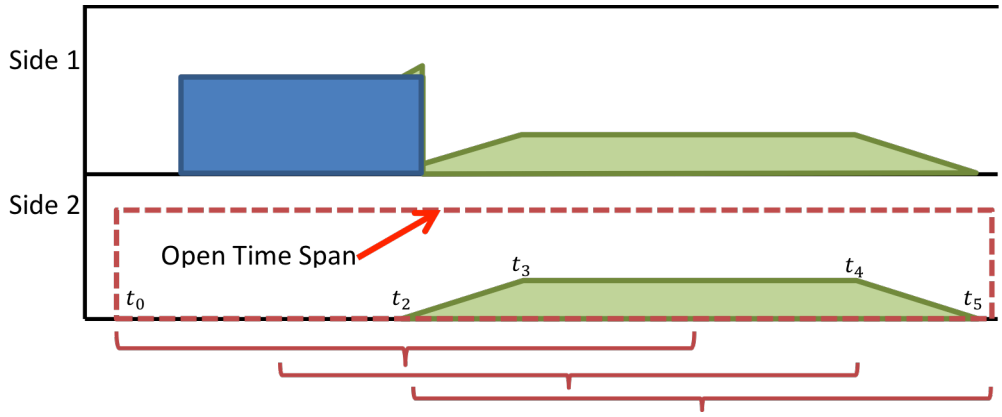

Fig. 10. Searching the red task's open time span for a temporal assignment that minimizes its bottleneck contribution. *The dotted box represents the entire range in which the red task can schedule, and each horizontal curly bracket represents a possible temporal assignment.*

With a high enough search granularity (e.g., one minute), it is probably sufficient to create a candidate time window for every possible start time within the open spans. However a simple optimization makes this search considerably more efficient: whenever the candidate window's start and end time both occur during a plateau in the bottleneck curve, we know that an iterative step one minute into the future will not improve the bottleneck result, and so we can "jump" forward to the next inflection point for either the start time or end time. In Fig. **10**, the first candidate window starts at t_0. Because the candidate window starts and ends on plateaus, we know that advancing to $t_0 + 1$ will not find a lower bottleneck peak. We can then advance the next candidate window until either its start or end is at an inflection point in the graph, in this case all the way forward until it ends at t_4 (the second horizontal bracket in the diagram). In this way, we can skip over many possible candidate windows and speed up the search process. Note that it is also possible, but much more complicated, to jump to the next candidate window when the start or end point of the current candidate window is on a diagonal[2]. For simplicity, we currently create a candidate window for every minute along a diagonal and have not seen a significant performance burden.

The following figures complete the example. The red task is allocated in the first time window (which has the minimum area under the bottleneck curve). Finally we consider the green task. Because it is the last task, it does not need to consider the potential allocation of any other task, so it either chooses its time window arbitrarily or uses

[2] Consider for example the case where the best possible window is in a valley with its start and end somewhere on the two opposing slopes. Determining the optimal location within the valley requires some clever arithmetic.

some secondary criteria. This was a very simplified example but should demonstrate Bottleneck Avoidance's ability to minimize conflicts, while considering each task only once.

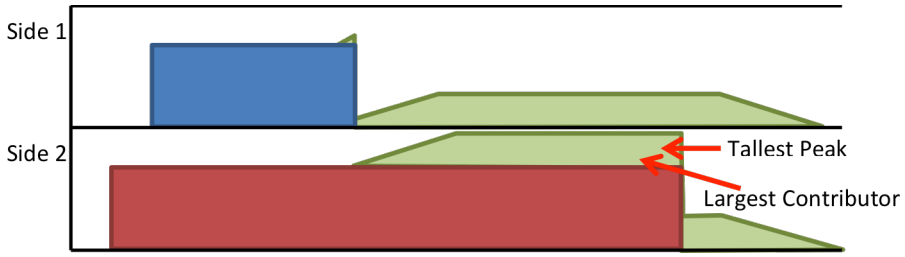

Fig. 11. Selecting the final task. *The red task has been scheduled in a time window that minimizes its contribution to the bottleneck curve. Now we select the final task for scheduling.*

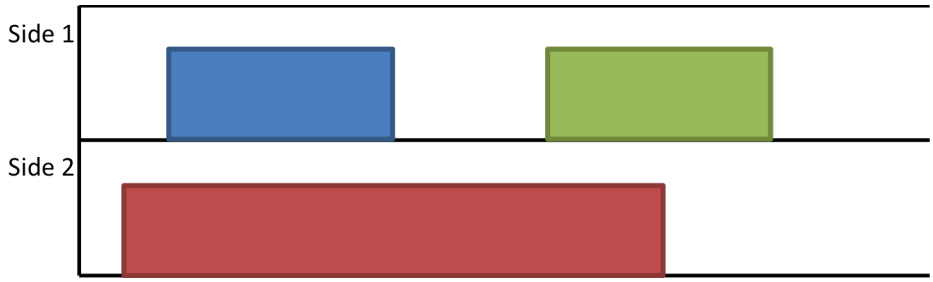

Fig. 12. All tasks are scheduled. *Bottleneck scheduling is complete with no conflicts.*

A Note about Meta-Resources and Shared Equipment

The discussion so far has assumed that different sides of a station use completely separate sets of equipment. This is true of most equipment (e.g., there is usually one antenna for each side); however, it is also possible for a single piece of equipment, like a communication channel, to be shared across several sides. Two tasks that use shared equipment can then conflict with each other *even if they schedule on different sides.* Rather than tracking bottleneck allocations for every piece of equipment in the system, we employ two types of meta-resources[3]: one representing all equipment at a side ("side meta-resource"), and one representing all shared equipment at a station ("station meta-resource"). All tasks have a possible requirement for each side meta-resource where they can schedule, and tasks that use shared equipment additionally have a requirement for each station meta-resource where they can schedule. By tracking bottlenecks on these two classes of meta-resource, we are effectively able to avoid conflicts on every piece of equipment without tracking each of them separately (which would be possible but costly).

6. RESULTS

We have obtained a sample set of 2847 tasks, representing a typical week's worth of requests. Using the tasks' default station/side and temporal assignment (ensuring that each has a valid visibility), 1846 of these tasks are scheduled in some type of equipment conflict. Using a reasonable set of assumptions about equipment alternatives and allowed changes from one side to another, Bottleneck Avoidance scheduling eliminated 74% of these conflicts, leaving 479 conflicted tasks. Examining a sample of the conflicts, all appear to have been truly unavoidable within the defined tasks' constraints (e.g., three LEO tasks with overlapping time windows on only two sides).

Computational performance of the algorithm is very good: on a modern, dual-core processor with 3GB of RAM, scheduling a given 8-hour period takes on the order of 3 - 5 seconds. Scheduling an entire week simultaneously takes on the order of 1 minute. Further optimization (of both memory and time) is certainly possible.

[3] Whereas most resources represent a real, physical piece of equipment, meta-resources represent a whole collection of equipment. These resources effectively have unlimited quantity – we do not track conflicts on them – but we do attempt to alleviate resource contention on each of them.

7. Business Rules

We were tasked with capturing the schedulers' experiential knowledge as to which constraints could be relaxed in what situations and to what extent. Initially, we planned to comb one year's worth of schedule databases and transaction logs, collecting instances where schedulers "bent the rules." We would then be able to programmatically build up a case-base of possible solutions, each with a prediction of how likely it was to be accepted by the requester. This eventually proved infeasible, primarily because neither code nor technical documentation on the format of the schedule database was available. It was suggested that we could instead capture these rules in a concrete, user-definable format – as a set of "business rules." This solution not only circumvented the technical obstacle, but it actually put more power in the hands of scheduling system administrators. The decisions of which constraints to relax are delicate ones, and the correct action in a given situation may alter over time as mission objectives or other factors change, so making the rules explicit and controllable ended up being a highly desirable side effect.

It was clear that what business rules should be applied and in what order varied among different groups of satellites, based on whether they we operated by the same organization or were essentially identical amongst themselves. So for convenience the business rules are stored in order of most preferred to least preferred with the group (often called family) of satellites they correspond to. The groups range in size from 1 to several dozen.

Business rules are generally applied after bottleneck scheduling is complete and only to the extent necessary to solve the conflict (e.g., if a prepass can shorten to a minimum of 5 min but only needs to shorten to 7 to resolve the conflicts it will shorten to 7). There are several ways to apply deconfliction strategies. The user has the option to apply business rules to a single task or to all conflicted tasks in the current time period. Rules are ordered such that those that cause the least "damage" are attempted first. Shortening a task's prepass is often one of the first steps a scheduler will take and with fairly accurate foreknowledge of what will and will not be an acceptable concession from the user. Because of this "shorten prepass" is among the first business rules attempted for many families.

Listed below are the parameterized prototypes for each single business rule deconfliction strategy that MIDAS currently supports. User-configurable parameters are represented in bold.

Shorten Prepass

- to a minimum of **(minimum_duration)** if **(condition)**
  - minimum_inter_family_duration = minimum duration if turning around from an Inter-Range Operating Number (IRON) in another family
  - minimum_intra_family_duration = minimum duration if turning around from an IRON in the same family
  - minimum_ats_duration = minimum turn around for an automated track support
  - hard_minimum = absolute minimum that will not be violated
  - condition = which minimum applies

Negative Turnaround

- allow a negative turnaround with a task in the set **(irons)**
  - irons = family of IRONs with which this task can have a negative-turnaround

Relax Schedule Constraints

- Remove NO# constraint
- Relax NO**(n)** constraint to a minimum of NO**(m)**
  - n = 2..∞
  - m = 1..n-1
- The NO(n) constraint requests that n supports in a row should not be at the same station

Redundant Equipment

- Remove **(secondary_equipment)** if **(primary_equipment)** is available
  - secondary_equipment = 1 or more equipment type
  - primary_equipment = 1 or more equipment type

Handoff

- Handoff task from **(source_stations)** to **(target_stations)** with minimum block size **(minimum_block)** and **(overlap_duration)** overlap
  - source_stations = allowed stations for the prior portion of the task
  - target_stations = allowed stations for the subsequent portion of the task
  - minimum_block = the minimum duration of each resultant task after splitting
  - overlap_duration = the desired amount of overlap between the prior and subsequent task

Move Out of Window

- Allow task to move **(start_change)** min earlier than the beginning of its requested window
  - start_change = maximum number of min before the window start
- Allow task to move **(end_change)** min later than the end of its requested window
  - end_change = maximum number of min after the window end
- Allow task to move +/- **(change)** min out of its requested window
  - change = maximum number of min outside of the requested window

Move to Another Rev (LEO-only)

- Move **(signum) (maximum_revs)** revolution **(condition)**
  - signum = [+ OR – OR +/-]
  - maximum_revs = maximum number of revs before or after the current rev that the task is allowed to move
  - condition = e.g., if a task with the same IRON does not already occupy this rev

Change Station (LEO-only)

- Move task from a station in **(from_stations)** to a station in **(to_stations)**
  - from_stations = allowed source stations
  - to_stations = allowed destination stations (provided the target station has an visibility that satisfies the request)

Shorten Task Duration

- Shorten task duration up to **(minimum_duration)**
  - minimum_duration = [### min OR ###% of original]
- Shorten task duration by moving start time **(maximum_start_change)** later
  - maximum_start_change = the number of min forward that the start can be moved
- Shorten task duration by moving end time **(maximum_end_change)** earlier
  - maximum_end_change = the number of min backward that the end can be moved

When applying business rules automatically to all conflicted tasks in the currently visible region, rather than applying all business rules to one task before moving on to the next task, we use an iterative algorithm to avoid highly damaging one task when a lower damage change to another task may have resolved the conflict. The algorithm, therefore, traverses all conflicted tasks, applying the lowest damage business rule first; then traverses a second time, applying the second lowest damage business rules until all conflicts are resolved or until we have tried all rules to all conflicted tasks.

There are situations when multiple business rules need to be applied to a single task. A single business rule may not be enough to resolve a conflict, or, by applying several business rules in concert to a small degree, we may be able to achieve a more desirable result than by only applying a single business rule to a greater extent. In general, two business rules are combined by applying the first rule to some maximal extent defined by the business rule, applying the second rule, and then relaxing the first rule. In practice, it is not possible to generalize the algorithm for combining any two business rules, and only certain combinations make sense. For this reason, we determined a manageable fixed set of combinations that are supported. Rule combinations are generally attempted after each of their component rules have been attempted singly.

8. BUSINESS RULES USE PROCESS

Once the business rules have been defined for each family of IRONs they can be employed in multiple ways and an initial deconfliction process has been developed that uses them in several ways. As mentioned previously, the first step is to apply bottleneck scheduling to solve conflicts by shuffling the scheduling within the parameters of each support request. Then, usually the next step is to automatically try to solve conflicts by applying a single business rule to each task, separately. Next, multiple business rules are applied to each task and each conflicting pair of tasks to solve each conflict while still relaxing the constraints for the least number of tasks possible and using the least damaging combination of rules for each task.

Even after following the above process, a fair number of conflicts usually remain. For hard to solve conflicts, human schedulers were observed "preparing" a location in advance of a move to solve a difficult conflict. For example, it may be the case that the conflict between two tasks cannot be resolved because although either or both of the tasks can be moved (within its constraints or using a business rule), all the possible destination locations are full. In this case a human scheduler will often work on one of these possible destination areas and move the supports that are there (either within their constraints or using business rules) to make room for one of the tasks in the original difficult conflict. So they move the other tasks first, then move the support into the hole they created. This is exactly equivalent to doing the same operations in the opposite order – first solving the original conflict by moving one of the tasks to a new location where there is not room for it and then resolving the new conflicts that were created. When viewed from this perspective, a domino phenomenon can be seen, resolving one conflict by creating another and then solving it. Said another way, moving one support forces another support to be moved. The above is a description of a single level of domino, but any number is possible and going to two, three, or four levels of dominoes is fairly common.

When using the domino method, the newly created conflicts, may be solvable with moves that are allowed by the constraints of the support, so that these should be tried first. But it is also often the case that solving the subsequent conflicts requires the use of business of rules too. Typically, using the dominoes method with a depth setting of 2, 3, or 4 is the last step in the automated deconfliction process. Human schedulers resolve the few remaining ones. The process of automatically employing business rules, in addition to mimicking the human deconfliction process, must also, since it is intended to perform within the current work flow, follow the human annotation process. This fulfills two functions. One is to make sure that the change is approved by the satellite operations center (SOC) before it appears in a published schedule. The other is to provide an audit trail of who made changes outside of the normal parameters. In the case of the automatic application of a business rule, the software did.

9. BUSINESS RULE RESULTS

The business rules were entered for the entire set 170 supported IRONs (actually about 2 dozen families) and applied to the deconfliction task for several different 24-hour schedules. Typically about 600 supports needed to be scheduled for each day. Although there were fairly wide variations about half of the supports started out in conflict. From this starting point of about 50% deconflicted, bottleneck scheduling typically solved half of the conflicts leaving 25% to be deconflicted by the processes described here. Applying single instances of business rules solved another 10% to arrive at an approximately 85% deconflicted schedule. Applying multiple business rules (but no dominos) generally brought that up to 90%. Applying the first version of dominoes using a depth of 4 brought that total up still further to around 95%. There are still several items left to be implemented so that we believe that we will be able to achieve a 98% deconflicted schedule.

10. CONCLUSION

Satellite Control Network scheduling largely consists of resolving disputes between competing support requests (and other tasks such as maintenance). About half of the conflicts can be solved by shuffling the requested supports within the constraints supplied with the requests. The other half require some degree of relaxation of the constraints. Using a representation of about 9 simple, parameterized business rules, families of IRONs, and a preference listing of the rules and parameters for each family allows software to automatically resolve the large majority of remaining conflicts. This greatly saves the labor required for deconfliction and allows more accurate study of future loading (and associated required resources) and more accurate response to what-if questions relating to the impact of failed resources and required emergency supports.

11. References

[1] Sadeh, N., "Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling," Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[2] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.