Bottleneck Avoidance Techniques for Automated Satellite Communication Scheduling

Richard Stottler[1] and Kyle Mahan[2]
Stottler Henke Associates, Inc., San Mateo, CA, 94404

Randy Jensen[3]
Stottler Henke Associates, Inc., San Mateo, CA, 94404

This paper describes Bottleneck Avoidance (BA), a one-pass scheduling technique, and its application to the domain of automated satellite communication scheduling. Optimal scheduling is in a class of computationally complex problems that require creative solutions to be solved in a reasonable amount of time. Satellite communication scheduling is particularly interesting because of a relatively high degree of resource contention and strong emphasis on scheduling *all* jobs without conflict. Generating a solution in this domain can be separated into two stages: first attempting to honor all constraints and requirements of the incoming requests, then negotiating with users to relax constraints and resolve unavoidable conflicts. The techniques and heuristics described here are intended to aid in the first phase. By carefully selecting the order in which tasks are assigned and then assigning them in a way that reduces resource contention, we can quickly generate a solution that attempts to minimize the number of conflicts and increase overall flexibility.

Nomenclature

BA	=	Bottleneck Avoidance
GEO	=	Geosynchronous Satellite
HEO	=	High-Earth Orbit or Highly Elliptical Orbit Satellite
LEO	=	Low-Earth Orbit Satellite
RTS	=	Remote Tracking Station

I. Introduction

THIS paper describes a one-pass scheduling technique – Bottleneck Avoidance (BA) – and its application to the domain of automated satellite communication scheduling. 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 Ref. 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.

[1] President, 951 Mariners Island Blvd, Suite 360, San Mateo, CA 94404, AIAA Member
[2] Software Engineer, 951 Mariners Island Blvd, Suite 360, San Mateo, CA 94404
[3] Group Manager, 951 Mariners Island Blvd, Suite 360, San Mateo, CA 94404

II. Motivation

Satellite communication scheduling can be seen as a two stage process: first attempting to honor all constraints and requirements of the incoming requests; then resolving unavoidable conflicts by negotiating with users to manipulate the constraints of the problematic requests ("bending the rules"). It is predictably advantageous to reduce conflicts as much as possible in the first phase, so that all or almost all conflicts remaining at the beginning of the second phase are truly unavoidable. This eliminates some of the guesswork and negotiation that goes into the rule-bending deconfliction phase. Because the initial scheduling follows strict and well-defined requirements, it is particularly amenable to an algorithmic solution (though there are certainly techniques that can be useful in the second phase as well).

Finding an optimal schedule for tasks across more than two resources is in the class of NP-complete problems[2], problems that quickly become intractable using classic techniques. Because searching the entire problem space for an optimal solution is infeasible, we relax this requirement and instead search for a "very good" solution, with a strong emphasis on scheduling all requests and meeting as many of the stated requirements as possible. We can apply creative heuristics that mirror human cognitive processes in order to generate a near-optimal solution with comparatively little computational effort, in a very reasonable amount of time.

III. Probabilistic Bottleneck Modeling

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).

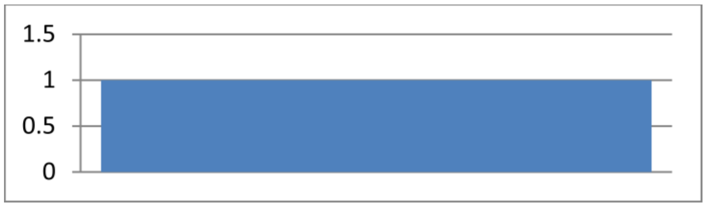

Figure 1. Predicted allocation of an inflexible task. *A task with one possible temporal allocation is represented by a block of predicted usage with quantity 1.*

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 Figure 1). On the other hand, tasks with several alternative resource assignments will have their contribution to the resources' predicted usages reduced accordingly. So if for example, the task in Figure 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 Figure 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:

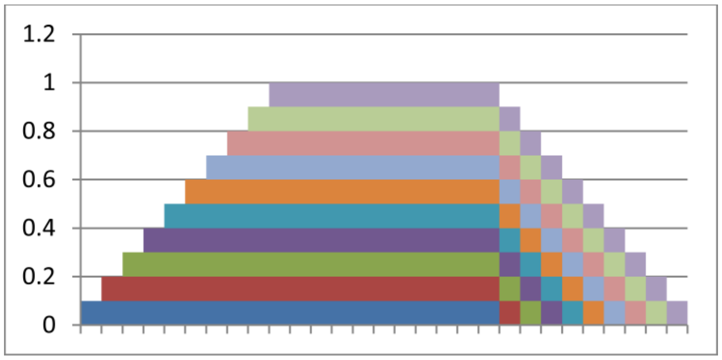

Figure 2. Calculating the predicted usage of a 20-minute task in a 30-minute window. *Allocate a 10% probability to each possible start time and sum them. Because every possible allocation includes the middle 10 minutes, the trapezoid plateaus at 1.0 — 100% probability.*

$$\frac{1}{2}(10\,minutes \times 100\%) + (10\,minutes \times 100\%) + \frac{1}{2}(10\,minutes \times 100\%) = 20\,minutes \times 100\%$$

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 Figure 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.

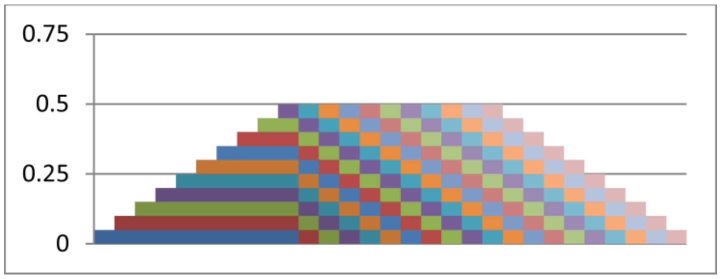

Figure 3. Calculating the predicted usage of a 10-minute task in a 30-minute window. *Allocate 5% to each of 20 possible assignments. In this case, the trapezoid plateaus at 0.5 – there is a 50% probability that any minute in the middle section will be allocated.*

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 Figure 4 and the following equations:

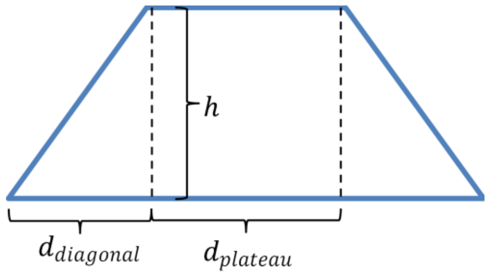

Figure 4. Trapezoid with labels.

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

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 Figure 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.

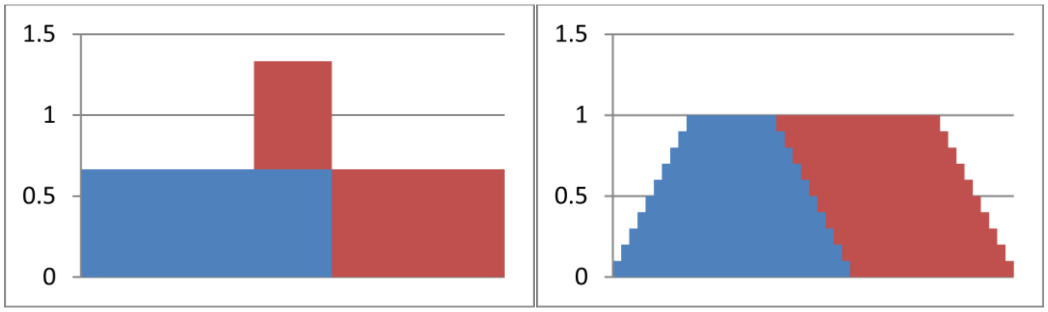

Figure 5. The predicted usage of two sequential tasks, approximated with rectangles (left) and fully calculated using trapezoids (right). *The first diagram incorrectly predicts a conflict between the two tasks, while the second reflects the fact that it is not possible for the two tasks to schedule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

IV. 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. Figure 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 Figure 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 Figure 2).

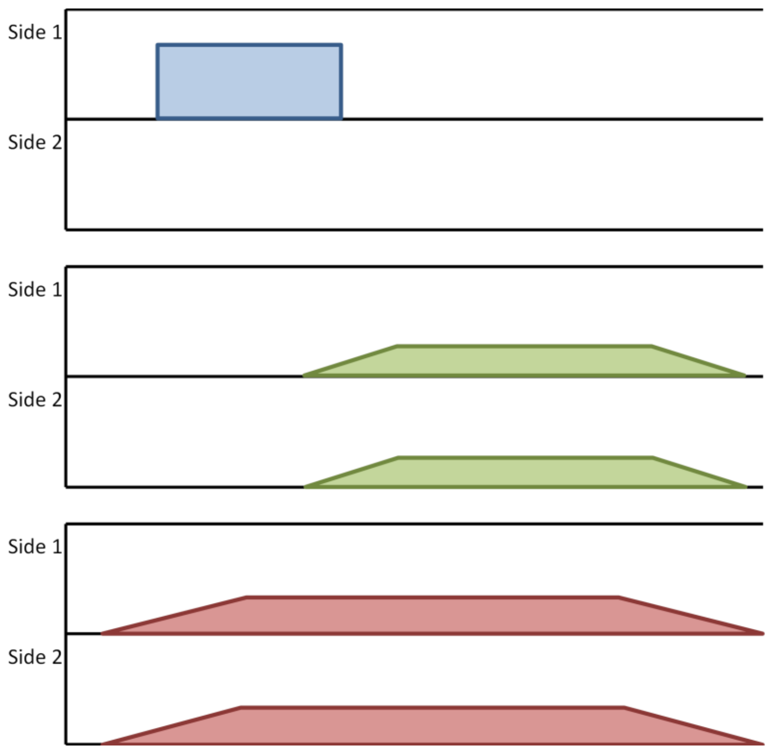

Figure 6. Predicted allocations for three tasks in a simplified scheduling problem with only two sides. *The blue box represents a temporally inflexible task. The green trapezoid represents a short, flexible task, and the red trapezoid represents a long, flexible task. The blue task can only schedule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

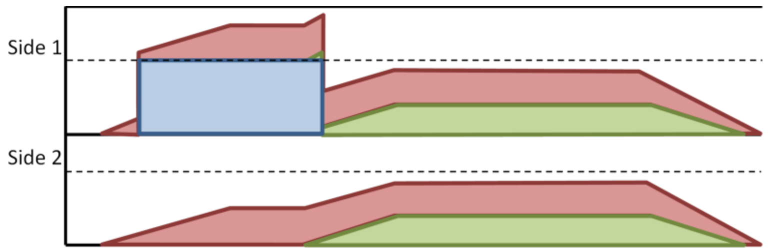

Figure 7. Aggregate predicted usage for the three tasks from Figure 6. *The dotted line represents a predicted usage of 1.*

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[*] 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.

[*] 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. Figure 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.

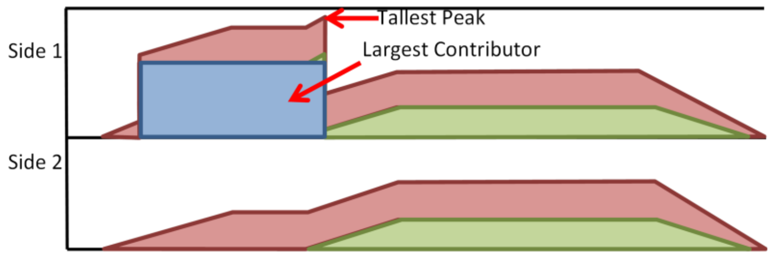

Figure 8. Selecting the first task to schedule. *BA looks for the largest unscheduled contributor to the tallest peak.*

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.

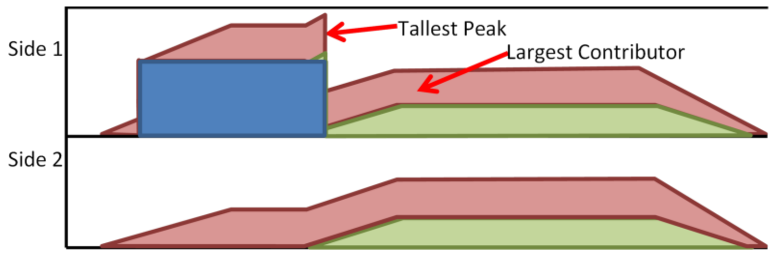

Figure 9. Selecting the next task to schedule. *The highest peak is the same as before, but now the red task is now the largest unscheduled contributor.*

Figure 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.

V. 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). Figure 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.

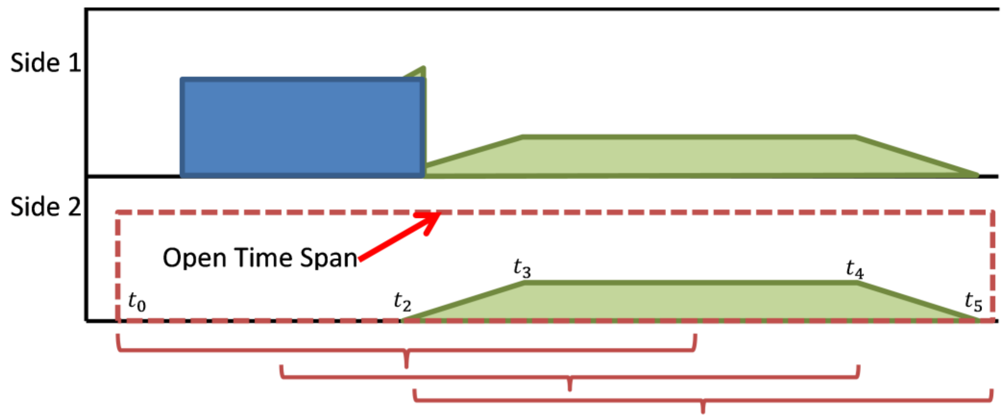

Figure 10. Searching the red task's open time span for a temporal assignment that minimizes its bottleneck contribution. *The dotted box represents the entire range in which the red task can schedule, and each horizontal curly bracket represents a possible temporal assignment.*

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 Figure 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[†]. 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

[†] 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.

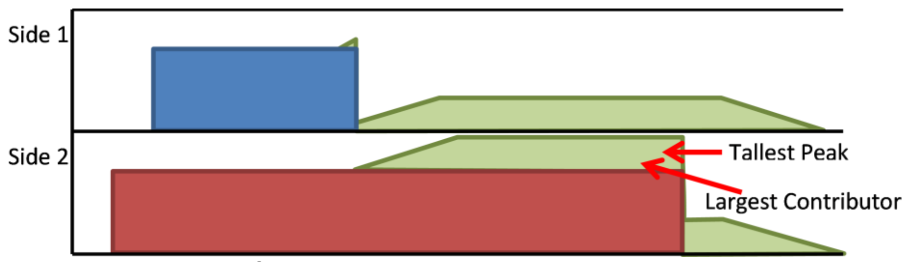

Figure 11. Selecting the final task. *The red task has been scheduled in a time window that minimizes its contribution to the bottleneck curve. Now we select the final task for scheduling.*

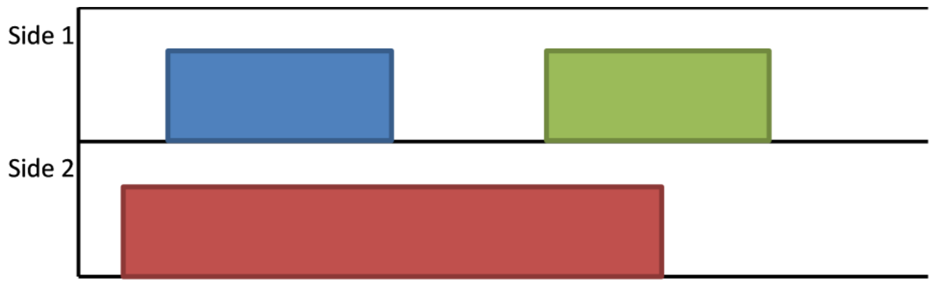

Figure 12. All tasks are scheduled. *Bottleneck scheduling is complete with no conflicts.*

A Note about Meta-Resources and Shared Equipment

Because it would be expensive and redundant to track resource contention on every piece of equipment individually, we create a meta-resource to embody all of the equipment at each side and track resource contention only for these meta-resources. In addition to equipment requirements, a task has a requirement for the meta-resource of each side where it can potentially schedule. This allows us to track and alleviate resource contention on each piece of equipment while only maintaining one bottleneck tracking model for the side as a whole. Note that we do not generate conflicts on meta-resources – if two tasks schedule simultaneously on a side, all conflicts generated will be over actual pieces of 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 conflict with each other *even if they schedule on different sides*. In our implementation, we employ a second type of meta-resource representing all shared equipment at a station. Tasks that use shared equipment have an additional requirement for each allowed station's meta-resource, and by tracking and avoiding bottlenecks on these meta-resources, we alleviate contention on shared equipment across an entire station.

VI. 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 remaining 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 5-10 seconds. Scheduling an entire week simultaneously takes on the order of 1-2 minutes. Further optimization (of both memory and time) is certainly possible.

VII. Conclusion

The expert schedulers who currently perform satellite communication scheduling are very good at their job. They routinely create elegant schedules that meet a large variety of user demands (both implicit and explicit), very rarely fail to resolve a conflict, and essentially never fail to publish a schedule on time. However, as the demand for satellite communications continues to expand, existing solutions, requiring *highly* experienced individuals, will be stretched further and further until they can no longer meet the demands placed on them. There is an opportunity here for automation to shoulder some of the burden of solving these complex problems. The Bottleneck Avoidance algorithm, using a probabilistic model of unscheduled requests, has been shown to quickly resolve a significant number of the conflicts that result from many users requesting contacts with many satellites. By its nature, BA alleviates unnecessary resource contention, keeping as much flexibility in the schedule as possible. This feature of the completed schedule will make even unavoidable conflicts easier to resolve. Combined with current efforts to automate the deconfliction of these unavoidable conflicts, it will soon be possible to generate a full day's schedule in minutes rather than hours. This will reduce some of the significant demands placed on human schedulers and help make the expansion of existing operations more feasible.

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.