Multi-Step Automated Deconfliction for Satellite Communications Scheduling

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

This paper presents a description of a method to allow software to automatically deconflict competing supports. It primarily consists of sets of business rules associated with sets of individual satellites that the rules apply to. Once defined, the business rules can be utilized in several different ways to achieve a 95% deconflicted solution as described in the results section. Planned future enhancements are also described. A 98% deconflicted schedule is expected.

Nomenclature

IRON	=	Inter-Range Operating Number
LEO	=	Low Earth Orbit
SOC	=	Satellite Operations Center

I. 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 a previous paper, we discussed a single-pass scheduling algorithm ("Bottleneck Scheduling")[1] 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 first step 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 discuss 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 rules (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.

[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, AIAA Contributor

II. 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.

III. 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.

IV. 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 soon be able to achieve a 98% deconflicted schedule.

V. Future Work

There are several logical next steps for this effort. There are improvements in both speed and intelligence possible with our current dominos software, including being more intelligent about which new conflicts to generate, the order to try them in, and the best use of bottleneck scheduling and business rules to solve newly generated, domino conflicts. Also directly related to automated deconfliction is going back to the original idea of using past precedent in the form of case-based reasoning (CBR) to make suggestions. These could potentially augment business rules when new resolution methods first start to be successfully used with an IRON family.

Business rules are defined in a well-known scripting language (Python) for maximum flexibility (i.e., so non-trivial properties and relationships can be reasoned upon when determining how and when a business rule can be applied). This format can be well understood by nonprogrammers with some experience and training. However, in the future it would be desirable to represent the rules in an even higher level domain-specific language that would be better suited for maintenance by a non-programmer. It would be possible to develop a relatively simple graphical editor for business rules with limited additional development effort.

A final very useful feature is termed by the users as “self-heal.” Sometimes the reason for applying a business rule has changed. E.g., it may be the case that to resolve a conflict, one of the supports involved in the conflict was moved outside of what would be allowed by its parameters so a business rule was invoked. If later the other support involved in the original conflict is deleted or moved for other reasons such that the moved support could be moved back to being within its original parameters, then this should automatically occur. (I.e., if the reason for applying a business rule changes and is no longer valid, the effect of the business rule should be reversed.)

VI. 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.

References

[1]Stottler, R., Mahan, K., and Jensen, R., “Bottleneck Avoidance Techniques for Automated Satellite Communication Scheduling,” *Proceedings of the Infotech@Aerospace 2011 Conference*, Vol.1, AIAA, Reston, VA, 2011.