A Flexible Architecture for Creating Scheduling Algorithms as used in STK Scheduler

W. A. Fisher and Ella Herz

Optwise Corporation and Orbit Logic Incorporated
fisher@optwise.com and ella.herz@orbitlogic.com

Abstract

A method of developing custom algorithms using the Optwise scheduling system is presented. An overview of the overall scheduling methodology is presented followed by a description of how schedule algorithms may be customized. Performance results for the commercial STK Scheduler regression tests used to verify algorithm coding enhancements and a case study are also provided.

Introduction

Scheduling a number of tasks to a pool of resources is a very common but difficult problem. For example in [1] Barbulescu et al, demonstrated that a generalized version of the range scheduling problem is NP complete. While this means that it may not be possible to construct an algorithm that can be proven to find "the" optimal solution in finite time for the general problem, real world problems can have specific conditions that make finding good solutions possible. Most successful scheduling algorithms used in the real world exploit special conditions that allow the users to find good solutions.

Orbit Logic and Optwise have found that a single algorithm for scheduling is not the best choice for all problems. By modifying various algorithm building blocks to meet both the desired solution goals and scheduling constraints, customized scheduling algorithms can be created that meet the needs of particular missions.

The purpose of this presentation is to describe how the Optwise scheduling architecture that is used in STK Scheduler can be adapted to specific scheduling problems. An introduction to the Optwise algorithms was presented at the 2004 STK Users Conference [2]. An overview of the custom algorithm interface was introduced at the 2008 STK Users Conference [3]. Some of that material is updated here for completeness.

Optwise and STK Scheduler

STK Scheduler provides the interface and top level business logic that translates a real world satellite task and resource problem into the specific language necessary for the Optwise algorithms to agnostically find a schedule solution.

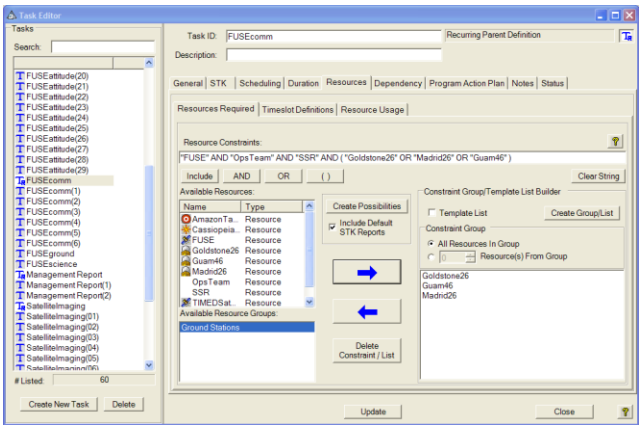

Figure 1: STK Scheduler Task Definition GUI

STK Scheduler provides the interface for defining the scheduling problem, configuring the algorithms, as well as displaying the results. The user defines resources and associated attributes and then defines tasks and their accompanying constraints and resource requirements and options. With a seamless interface to Systems Tool Kit (STK) the user can apply physical constraints to the scheduling problem by selecting one or more STK computations as resource availability or task scheduling constraints. The software then uses the Optwise algorithms to find de-conflicted, validated schedule solutions which can be viewed in table, Gantt, or 3D map views. STK

Scheduler provides both a graphical and application programming interface (API) for defining and solving scheduling problems and exporting the results.

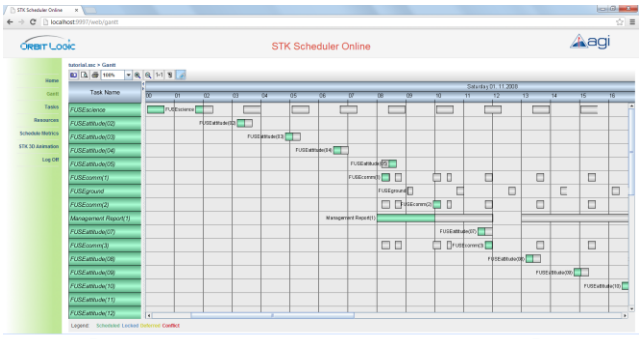

Figure 2: STK Scheduler Gantt View

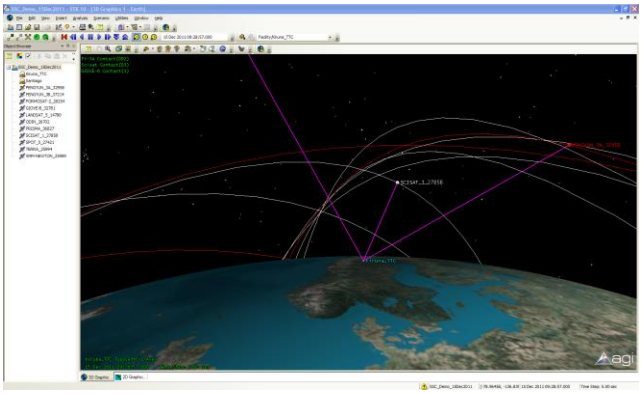

Figure 3: STK 3D Schedule Animation

Within this paradigm, the user can create and modify plans manually or can employ one or many algorithms to solve the scheduling challenge. The user can also utilize the custom algorithm builder to define an algorithm specific to their scheduling problem. Regardless of the algorithm applied, a user may edit the resulting solution if desired.

Over 200 licenses of STK Scheduler have been deployed internationaly supporting commercial, civil, and military programs. The user list includes Boeing, Harris, Raytheon, Northrop Grumman, Booz Allen, Lockheed Martin, Honeywell, General Dynamics, Orbital Sciences, TASC, Scitor, EADS, NASA. Pprograms supported by STK Scheduler include the Air Force Satellite Control Network (AFSCN), SBIRS, SBSS, STSS, OSIRIS-REX, GLAST, WIRE, NASA Ground Network, and NFIRE. In addition, Orbital Sciences uses STK Scheduler for all launch and early orbit planning for all of their space launch vehicles (Taurus, Antares, Pegasus, Minotaur).

The STK Scheduler Model.

In order to understand its custom algorithms, it is important first to understand the STK Scheduler model and how Optwise Algorithms fit into that model.

In the Optwise scheduler model tasks are assigned to time slots. As shown in figure 4a) each time slot has an earliest start and a latest stop. The desired task duration is assigned during this time slot. The time slot also has associated with it a profile. The profile is a container that allows one or a combination of resources to be defined. It is often the case that a task may be satisfied by more than one profile as shown in figure 4 b).

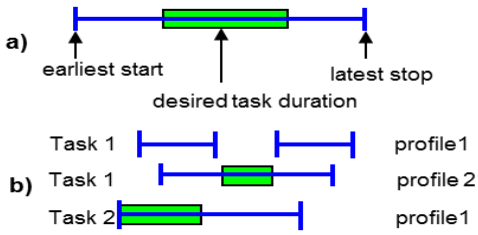

Figure 4: Tasks and profiles

For example profile 1 might be the combination of satellite A and ground station A and profile 2 is the combination of satellite A and ground station B.

In STK Scheduler the solution profiles are derived from real world resource constraints, such as line-of-site access from a satellite to a ground station, or limited satellite onboard memory, or the times when an operator is available in the control center.

STK Scheduler uses STK to model physical constraints, and provides a means for non-computed constraints to be defined as well. All task and resource constraints and computations are used to generate windows of opportunity that could be used to accomplish the task for a given resource profile. In STK Scheduler these windows are called “timeslots”. The process of creating the feasible timeslots can be simple (a point-to-point STK access calculation) or complex (multiple calls to STK access calculations with internal STK constraints combined with external user-defined generated constraints). Any number of resources may be specified in a task profile definition using Boolean logic to allow a variety of resource combination options for any task. In STK Scheduler each valid resource combination is called a “profile”. STK Scheduler then passes to Optwise a number of profile possibilities including associated computed timeslots. Each profile and timeslot also has a desirability value associated with it. In STK Scheduler this is used to include user resource and time preferences (for instance, it may be more desirable to schedule the task in an early timeslot with a specific resource profile).

Tasks may also have several other proprieties including priority, desired start time, and duration types. Tasks can have fixed or variable durations. Variable duration tasks will only be assigned if a minimum duration can be assigned, but will be expanded up to a maximum duration

if possible. Tasks can also be defined to only be allowed to use one slot or to use multiple slots. When multiple slots are allowed the user also has the option to allow the task to switch resources between assignments if needed to achieve the desired total assignment duration.

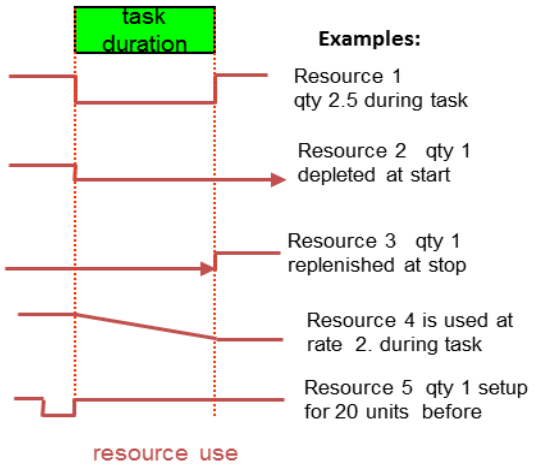

Figure 5: Resource Usage

As shown in figure 5, the resources in a profile may be used during, allotted at the start, or replenished at the end of tasks. It is also possible define a resource to be use or replenish at a rate. For example a particular task may deplete a battery resource at particular rate, and another task may charge the battery at different rate. A setup resource may be defined for use prior to the timeslot access. Note that this resource is used outside the time slot time. It is also possible to define a breakdown time. All resource levels and time values are computed with double numeric precision. All of these properties are defined for each profile.

It is also possible to define task to task constraints such as task A must be some amount of time after task B. One can also define a minimum and maximum between tasks that have been define to be in a group.

Model Figure of Merit

The problem input language also allows the user to adjust parameters in the figure of merit that is used to measure the "goodness" of a solution.

$$FOM = \sum_{task=i} \mathrm{Pri}_i \begin{pmatrix} K_{assign}(Ua_i) + K_{dur}(AssignDur_i) + K_{desire}(SlotDesire_i) \\ + K_{early}(EarlyBonus_i) + K_{max}(MaxBonus_i) \\ + K_{userstart}(|DesiredStart_i - Start_i|) \end{pmatrix}$$

$$+ UserFunc$$

$$Ua_i = 1: assigned, \quad 0: unassigned$$

K_{assign} is used to adjust the relative weight for pure assignment. This term is complementary to the next term.

K_{dur} is used to adjust the relative weight of assignment times the duration. Thus tasks with longer durations will affect the FOM more than shorter ones.

K_{desire} is used to adjust the relative weight of desirability of the slot assigned to the task.

K_{early} is used to adjust the relative weight of the early bonus. The early bonus is 1.0 if the task was scheduled as early as possible and zero if as late as possible for that task.

$K_{MaxBonus}$ is used to adjust the relative weight of the Max duration bonus. It is 1.0 if maximum duration is scheduled and 0 if minimum duration is scheduled.

$K_{userstart}$ is used to adjust the relative weight of the userstart bonus. The userstart bonus is 1.0 if the task was scheduled as close to the desired user start and zero if far away as possible for that task.

UserFunc is a link to external user function. (dll)

The user is able to change the values of K values in the problem definition file in order to adjust how much each term contributes to the FOM.

Algorithm Heuristic Classes

Before going into the details of the algorithm heuristics it is useful to take a wider view. The algorithms used fall into two major classes, ordered algorithms that assign tasks by trying time slots in some order and the neural algorithm which uses a competition model.

In the ordered type of algorithm, time slots (and thus the corresponding tasks) are tried in some order. As time slots are used for a task assignment they may block future assignments. In the left top panel of figure 6 just such a case has occurred (It is assumed that the task requires all capacity of a resource). The slot "consideration" order is shown by the numbers near the end of the slot. By going in a different order, two other solutions result in fully assigned schedules.

In Figure 6, three examples using different orders with one pass are shown. In the top example only one task is assigned because the assignment of task1 to slot 1 blocks the only available slot for task 2. The next two examples show slot checking orders that result in full assignment.

It is also possible to use information during the process to modify this slot checking order. Multiple passes can be used to add additional tasks that may not be schedule in a single pass. For example, assume the last task to be considered (and assigned) is one which replenishes a resource. A second pass might allow a task unassigned during the initial pass to be assigned to the replenished resource. Repair heuristics are implemented by removing

a task from a "finished schedule" and then testing if other tasks (of higher value) may be done. A Greedy algorithm can be implemented by selecting the next slot based on its possible improvement to a Figure of Merit. Which detailed strategy is best depends on the specific problem.

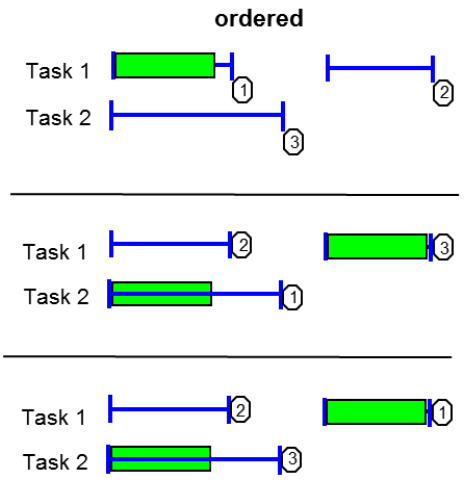

Figure 6: Time slots consideration order

We will return to the details of how the various ordered search algorithms can be constructed after a discussion of the other class of algorithms.

Figure 7 shows an example of a second class of algorithm used in the solver engine. This method evolved from an analog neural network approach for the task assignment problem suggested by Kennedy and Chua, 1998 [5] and investigated in [6]. We again begin with a model based on time slots.

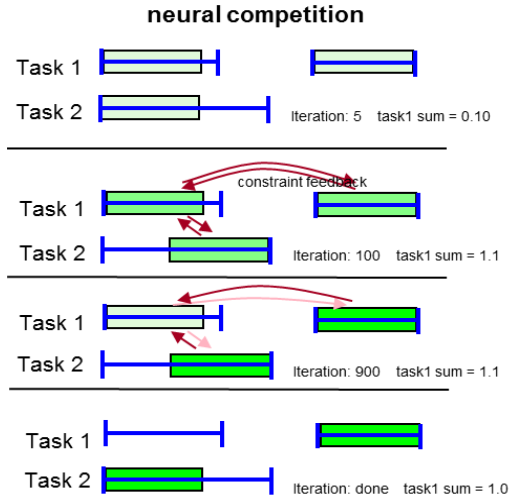

Figure 7: Neural Competition

Each time slot represents a possible way to "assign" the task and has an assignment probability. All time slots start with an assignment probability near zero. Each time slot probability is allowed to grow subject to constraints. An example of one fundamental constraint is: the sum of the probabilities of two overlapping assignments must be less or equal to one. The algorithm is designed to shift the start times of the two tasks in conflict and to reduce the probabilities to resolve the conflict. The "assign the task only once" constraint is creating by requiring the sum of probabilities for all time slots for a particular task to be less than or equal to one. If the sum is greater than one then all time slots in the sum are penalized. If there is a resource capacity violation at a particular time then the time slots probabilities are penalized and the start times of the tasks are adjusted to help clear the constraint. This is done for all time slots simultaneously.

Figure 7 shows this process for a two task problem. Note that in the middle panel (at a hypothetical iteration 100) the first timeslot probability for task one is penalized twice while the other two time slots probabilities are penalized once. These two time slots will "outgrow" the doubly penalized timeslot eventually forcing the first time slot of task 1to a zero value. This allows task 2 the freedom to move earlier in the schedule. The order of the time slots is no longer a factor. The solution is driven by the problem constraints. The solution represents a maximization of an underlying objective function which is to maximize the number of assignments. In the model implemented for the scheduling algorithm engine the objective function maximizes total assigned duration.

Although the example above goes to the optimal solution, it is not the case if slot 1 is given an initial value significantly larger than slots 2 and 3. This illustrates another interesting feature of using such an algorithm. By varying the values of the initial nodes randomly, it is possible to generate a family of solutions. Notice that unlike an ordered search algorithm there is no task or resource order rules; the constraints are feedback loops that constrain the solution space.

Regression Tests Results

Optwise and Orbit Logic maintain a library of over two hundred past problems that test the feature space of the scheduler. The regression tests demonstrate the complexity and speed with which STK Scheduler and various algorithms solve these problems. By running a variety of algorithms on a variety of problems, it is clear that one algorithm is not best for all problems.

Most of the regression tests contain less than 100 tasks and solve in less than a second with most of the algorithms. They are important, but yield little information for evaluating performance. Here we discuss four of the most complex tests from a computation point of view and two smaller test problems. All solution times in this section were benchmarked on a 64 bit Intel Core 2 2.13 GHz processor.

The names have been replaced in the four customer derived cases. Table 1 gives an overview of the dimension

of the problems with regard to the number of tasks, resources and timeslots that are covered.

name	#tasks	#resources	#slots
15T2R	15	2	20
OL-Ex10	120	80	363
Cust1	6510	11	6720
Cust2	619	163	26850
Cust3	553	160	24580
Cust4	526	58	4006

Table 1: Regression Test Dimensions

The 15T2R test was designed to have many tradeoffs involving tasks resource conflicts. With this setup, the Neural algorithm performs very well. However, despite what was designed to be a stressing problem, the Random algorithm with one trial found a full assignment solution. A more detailed analysis in [7] showed that the probability of getting a full solution with one run of Neural was 72 % and with one run of random it was 48 %. All cases took less than a second to run. The FOM was programmed to reward total task duration and earliest start. For fully assigned solutions higher FOM measures how early the tasks were scheduled.

Algorithm	# assigned	FOM
OPS	13	221.8
Sequential	13	221.8
MPS	14	230.3
Neural(1)	15	243.3
Neural(100)	15	243.8
Random(1)	15	238.5
Random(100)	15	244.2

Table 2: 15T2R Regression Test Results

The OL-Ex10 test contains 120 tasks and 30 resources using four task types with varying fixed duration times. STK accesses were obtained for 10 hours assuming simultaneous access to a ground station and some elevation constraints. By using a 10 hour time period, the total resource availability was reduced to a point where the scheduling algorithms needed to do some real work. This problem is an example of one that is solved well by the Sequential algorithm. Only the Neural algorithm was capable of getting a full solution prior to the invention of the Sequential algorithm. Notice that the Sequential algorithm has the highest FOM score. That is because it has the tightest packing of the tasks.

Algorithm	# assigned	FOM
OPS	116	195.9
Sequential	120	239.1
MPS	116	218.2
Neural(1)	120	202.1
Neural(100)	120	204.0
Random(1)	120	196.0
Random(100)	120	199.8

Table 3: OL-ex10 Regression Test Results

The next four examples are all large problems derived from customer problems. All examples include variable duration tasks which can be expanded, but have limited or no handover tasks. They are 1 or 2 day schedule periods with minimum durations as small as 2 minutes and maximum duration ranging from 5 minutes to unlimited. In the regression test each of the standard algorithms is run. The Neural and Random algorithms are run with number of tries of 1 and 5. Table 4 shows the number of assignments for each customer derived case with each type of algorithm.

Case	OPS	Seq	MPS	Neural (1)	Neural (5)	Ran (1)	Ran (5)
1	37	37	37	44	44	48	53
2	577	598	615	573	578	607	609
3	544	540	550	526	527	543	549
4	255	251	267	259	263	254	253

Table 4: Customer Regression Test Results (Assignments)

Notice that no one algorithm stands out as best for these problems. The Random algorithm with 5 tries appears to be the best. The MPS method is next in terms of performance. Note here that performance is analyzed based on the number of assigned tasks. This is not unusual when interacting with the STK Scheduler customer base. More tasks seem always to trump even the best constructed FOM score. Based on customer feedback it is clear that customers place a premium on predictability. Unless one of the algorithms has found a solution with a much higher assignment percentage they will use one pass or sequential. Both these algorithms were designed to follow typical human strategy.

Case	OPS	Seq	MPS	Neural (1)	Neural (5)	Ran (1)	Ran (5)
1	13	13	31	15	26	14	15
2	9	8	13	171	823	8	11
3	8	7	12	170	818	7	9
4	2	2	2	7	28	2	2

Table 5: Customer Regression Test Results (Solution Time)

The most interesting thing to note in the time to solve metric is that the Neural algorithm takes significantly longer than the rest for large problems. The Neural algorithm must set up a feedback network based on the number of timeslots. In the case 2 and case 3 that is 26850 and 24580 timeslots, respectively. In this problem most those nodes have no interaction with most of the other

nodes. Because these problems have weak interactions between the resources most of the neural computation is summing zeros. In this case, the Neural algorithm is a bad choice.

While not yet in the regression test but under current development, we have done one problem which had a request for 29,745 tasks to be assigned over a 24 hour period using 33 resources. There were an astounding 1,710,462 possible accesses (time slots). Using a modified version of the input routine and a custom algorithm very similar to the Sequential algorithm (no expand) we were able to find a solution with 20,706 assignments in 161 seconds. That is over 120 assignments per second. This is an example a simple but extremely large problem. It illustrates efficiency of the underlying code and the ability of the algorithm to handle large problems.

The results of these regression tests show that across a wide variety of scheduling problems with different performance criteria, there is no single "best" in the "standard" set of algorithm.

Solver Architecture and Custom Algorithms

Next, we turn our attention to the custom algorithm feature. The methodology that is used to find a "schedule solution" is embedded in a solver engine that has been designed from the ground up to be adaptable. It uses a run time interpreted script language to define the algorithm.

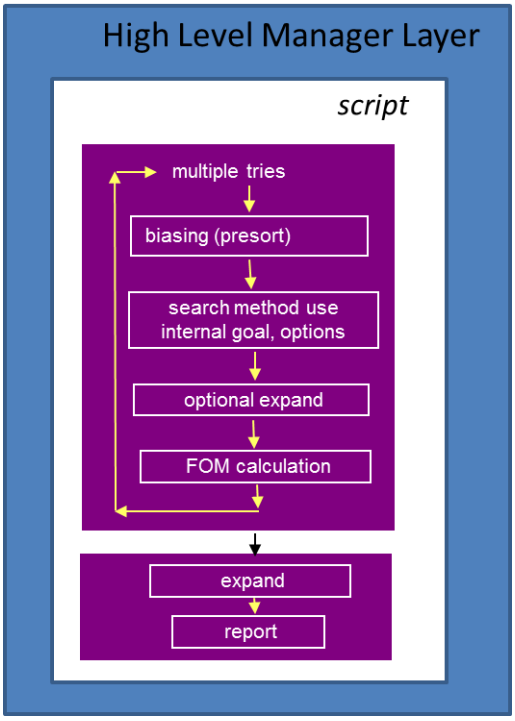

Figure 8: High Level Manager

The job of the high level manager shown in figure 8 is to parse the input problem description and input solution script, track solutions generated by the solution script and provide output back to the client program (In this case STK Scheduler).

The high level manager is also the place in the code where "strategy" is added. Examples of such strategy will emerge later in the discussion.

Each box indicates the logical flow of a typical problem. The scripting language parallels this flow. These blocks also roughly indicate the independence of the underlying computation. The code in some of these routines has been optimized over the past 10 years to allow near real time turnaround for many typical problems. Each component has also been designed to be modified.

The basic structure of a solution script is shown in figure 9. A solution string is a xml format file that contains one or more trial specification. A trial specification contains steps and modifiers. Steps are order specific, modifiers are not. At a minimum, a trial must have a search step. All other parameters are optional. Typically a script that uses an ordered search will also have a sort step.

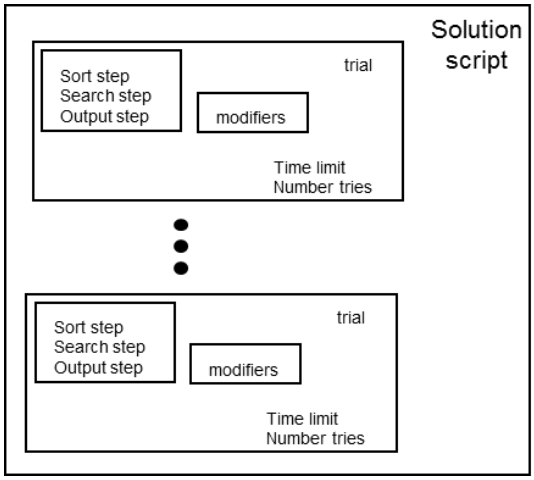

Figure 9: Solution Script Structure

The architecture allows up to three sort levels, each with a parameter that defines the preference for sorting. Sort parameters can be set to list, priority, early start, desirability, slack, or random. For list, priority, and desirability, the sort parameter can be set to ascending or descending. For random sorts, the randomization can be by task, profile, or time slot. A randomizer allows a random choice from remaining slots if all the prior sort criteria are equal. The randomizer name determines whether the randomization will occur over all slots, slots associated with tasks or slots associated with profiles. Use of a randomizer instructs the manger to loop over the trial by the named variable number of times, randomizing the final variable. This allows a Monte-Carlo like search

strategy to be implemented. The solution with the best figure of merit is chosen.

Multiple modifiers can be selected and are not order dependent. The first modifier is the use of Priority Groups. If the Priority Groups modifier is selected, slots are first placed into as many groups as there are unique priorities. This allows multiple passes to be made over the slots in the group to obtain maximum group assignment before moving on to another priority group. Whether or not priority groups have been chosen as an option, a single or multiple dimensional sort of the slots is available.

The next two modifiers deal with the Figure of Merit. With a Greedy Figure of Merit modifier enabled, each time an assignment attempt is completed, the Figure-of-Merit (FOM) is checked for all possible assignments assuming success. The slot with the largest possible increase in FOM is checked next. This mimics a classic greedy heuristic but with a user definable FOM. The External Figure-of-Merit modifier directs the engine to call an external method within a user modifiable dll to calculate the Figure of Method (FOM).

There are a number of expand modifiers because expand is a computationally expensive operation and expanding a task early in the assignment process will tend to block later assignments. Having the ability to first assign the minimum duration required to meet the task requirement and later expand duration if it is possible has resulted in ability to have maximum task assignment and extremely high resource usage. Expand modifiers are: Expand after Each Try, Expand After Each Task, Expand Handover Task Up Front, and pre-Assign and Expand Soft Tasks.

The pre-Assign and Expand Soft Resources modifier instructs the manager to find tasks that use soft resources and assign them first. Soft resources are typically rate resource that are consumed by some data tasks and replenished by other. The resource also has a soft limit to how much they can be replenished. A typical example is a battery recharge operation from a solar collector. When the battery is full the charging operation is turned off automatically at its limit. The easiest way to model it is to have a task that is always on whenever there is sun access. The onboard system knows when the battery is full and soft resources fulfill the same function. Finding these tasks and making sure they are on and fully expanded always improves the changes of getting tasks that will require the resource on.

There are three possible search options, ordered, neural and DS1MPS. No sort step is required for the neural or DS1MPS. DS1MPS implements code that mimics the performance of the original algorithms. It requires the manager to set up some special tracking since internally it is equivalent to doing multiple trials with different presorts.

The final step option is an output step. While it seems this is essential, for STK Scheduler it is not needed because STK Scheduler is able to read the solution string from the Optwise scheduling object directly without file I/O. You might want an output step if you wish to have an additional file written at the end of a series of trials. The file format can be a traditional flat file or xml and you can select from the last completed try from a trial (current) or the best solution found thus far.

Building Custom Algorithms

While it is possible to create the custom scripts by hand editing, a GUI Algorithm Builder is available from within STK Scheduler. The user can simply select the sort parameters from a list, and chose behavior modifiers from check boxes. A screenshot is shown in figure 10.

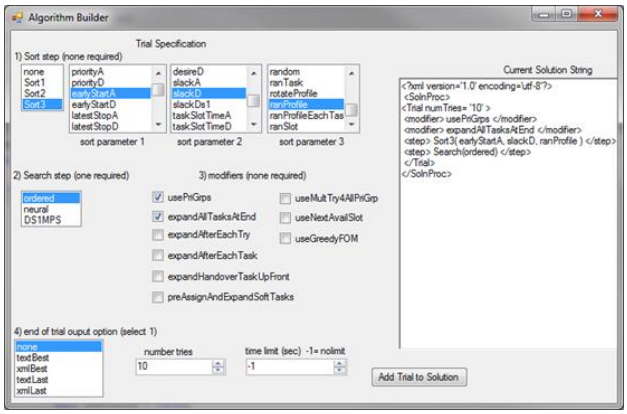

Figure 10: STK Scheduler Custom Algorithm Builder GUI

The process is simple. Select a 1 2 or 3 dimensional sort, use pull down list to select a sort parameter (an A or D at the end of the parameter names means ascending or descending sort), select any modifiers that are needed, if a randomizer or neural is used select the number of tries to use, select a maximum time to allow for a search, and select a output type if needed. When the add Trial to Solution button is press an xml string is added to the existing solution. You can have multiple trials in a solution.

Example Solution Strings

The solution string is a simple xml format string. Each solution string can have one or more trials. If there is more than one trial than the manager keeps track of the best solution based on the FOM. Note: the order of modifiers is not important but step order is. A search is the only required step. Slots will remain in the order they were listed in the input description file if the sort step is omitted. It is possible to output a name value formatted or xml formatted outfile at the end of each trial. The interface of the software module also allows the calling client to retrieve the solution data without file I/O. For example the

solution string that corresponds to the One Pass Algorithm in STK Scheduler is:

<?xml version='1.0'encoding=utf-8'?>
<SolnProc> <Trial>
<modifier> usePriGrps </modifier>
<modifier> expandAllTasksAtEnd </modifier>
<step> Sort1(desireA) </step>
<step> Search(ordered) </step>
</Trial> </SolnProc>

In the One Pass algorithm task are passed into priority groups and sorted by slot desirability. If a slot has an identical priority and slot desirability then the list order in the input description is used. Thus, if there are a number of tasks with equal priority and all slots have equal desirability, then the first task and slot in the input file list order will be tried. Since the slots associated with a task tend to be written in a group typically the one pass algorithm will search all slots for a particular task before moving on to the next task. It is not a requirement that all slots for a task be written at one time, it is a tendency of most users. In STK Scheduler the desire parameter is used to adjust slot desirability based on priority of the resources in the profile and scheduling preferences. If default values are chosen then it is the same for all time slots. It is however another way to adjust the algorithm performance.

Making one change to the string; replacing "Sort1(desireA)" to"Sort1(earlyStartA)" changes the algorithm to match the Sequential Algorithm in the standard algorithms listed in STK Scheduler. In the sequential algorithm slots are again placed into priority groups. But now since there is a sort(EarlyStartA) slots are sorted by their slot start time in ascending order. Thus, the task with the earliest possible slot start will be tried first. If the task assignment is blocked, then whichever task that has the next earliest slot time will be tried. What happens if there are multiple slots with the same time? The list order of the slots is used. This algorithm is a direct mapping of a customer request. Many problems can be solved with this first available approach and it tends to match a strategy that human schedulers are comfortable with.

The string for the MultiPass Algorithm is very simple: <Trial> <step> Search(DS1MPS) </step> <modifier> expandAllTasksAtEnd </modifier> </Trial> . The multi-pass algorithm was a first generation algorithm that used multiple tries with different presorts to find the best solution. Depending on the number of resources in the problem it tries up to 32 different presorts including a phase in which slots are randomize by profile. In order to be able to exactly match the original algorithm performance, any step based sort parameters are ignored. MultiPass inspired the custom algorithm approach. Rather than having the presort selection rules fixed in logic, it is now possible for the end user to create their own. Something similar to MPS could be created in a custom algorithm by using multiple trials.

The string neural algorithm is very similar to the MPS string: <Trial numTries= 5> <step> Search(DS1MPS) </step> <modifier> expandAllTasksAtEnd </modifier> </Trial>. As described above the neural algorithm does not search in any order so a sort step is ignored. Notice that a new variable is now defined in the trial header, numTries=5. This instructs the engine to do 5 runs of the neural scheduler using random starting conditions each time. Again the best solution based on the FOM is saved. Neural works best on problems in which many tasks using the same resources at nearly the same times.

The final algorithm in the standard set of algorithms is STK Scheduler is Random. It was originally written to be a base case for judging performance. It turned out to be a great algorithm for problems where there were multiple similar resources and uniform usage was desired. Such resources tended to have similar but not quite identical access times. The solution string for random is <Trial numTries=30> <modifier> expandAllTasksAtEnd </modifier> <step> Sort1(ranDS1) </step> <step> Search(ordered) </step> </Trial>. The key difference is that there is one sort parameter, ranDS1, which is a special version of the random sort. ranDS1 exactly matches the way that slots were randomized in the first generation "random" algorithm. It is most similar to ranSlot. The difference is in how slots are randomized. There are now many more options for controlling how much a when randomization is applied.

Randomization of the final variable is one example of a scheduling strategy that is added at the manager level of the software. The manager simply makes multiple loops over a trial randomizing the variable and checking the FOM. Repair as described in [8] [9] can also be implemented by modifying this level of coding. For repair the high level manager uses some method to select a portion or portions of a schedule solution to freeze and then schedules around the partial solution. The routines that expand the task duration at the end of the schedule are an example of repair behavior. Those routines are clever enough to be able to move a task later in time that is constraining the duration expansion of a primary task.

NASA Case Study Example

Next consider an example of a problem we investigated as part of a proposal to demonstrate the application of STK Scheduler to the NASA Ground Station Problem. For the problem we modeled a two week period during which 41 spacecraft with multiple view requests were assigned to one of 20 ground station antennas. Since each spacecraft mission was assigned a unique priority, there were 41 priorities levels. Based on desired the number of desired

events per mission and the availability of the resources the feasibility modeling of STK Scheduler generated 4452 possible tasks and 158,069 time slots.

The customer also defined two goals. The primary goal was to assign the largest number of events to the highest priority missions first if possible. The secondary goal was to maximize assignment and match target resource usage as define in an external spreadsheet. The purpose of the spreadsheet calculation was to penalize solutions in which an individual resource did not match contractually obligated resource usage levels for some resources. Over usage incurred additional charges while under usage implied wasted contract time. The spread sheet calculation computed a composite which tried to capture these two goals but did not strictly enforce the first goal.
The proposed figure of merit was

FOM = 1 + (100* A -1000*B - C)/D, where

A = ∑(number of events over desired),
B = ∑(number of events under desired),
C = ∑(individual resource usage penalty) , and
D = ∑((501-pri)*number desired events).

The sums are over the missions except for C which is over the resources. For the STK Scheduler modeling of this problem, the number of tasks is equal to the number of desired events. It is not possible to assign more than the desired number of events and A will always be zero. The resulting FOM in form is 1- ∑ δ/K - C/K. (δ= 1 if a task is assigned and 0 if not). K is a new constant that correctly scales between the reduced form and the correct spreadsheet formula. The key point is that ∑ δ is an available term in the existing FOM. The C/K (penalty) term must be computed externally. Figure 11 shows data being exported from STK Scheduler to and Excel macro for calculation of the NASA FOM.

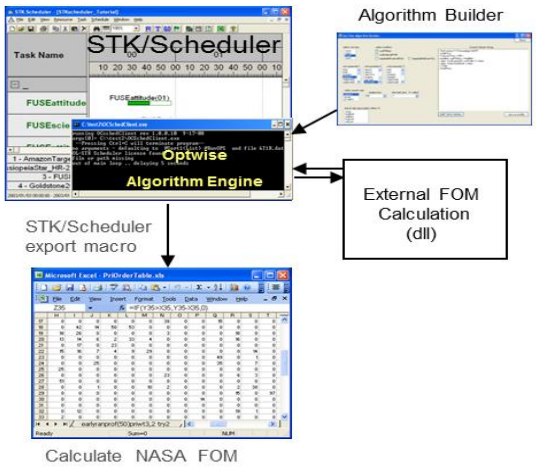

Figure 11:NASA Case Study Architecture

The first test was to try the existing standard algorithms and ignore the penalty term and use data exported from STK to calculate the FOM. The best standard algorithm was the Sequential Algorithm.

The second test was to use Algorithm Builder to create a custom algorithm again ignoring the penalty term. The best custom algorithm found was to add randomize profiles to the sequential algorithm with 50 tries.
For the third test an external FOM was coded that was used to select the best solution from the multiple tries.

Finally, the greedy option with the external FOM was tried which checks the delta FOM of all possible slots to choose the next slot to try. The external NASA FOM had to be modified slightly to allow a better delta FOM to be calculated.

Table 6 shows how each algorithm fared with regard to number of assignments, FOM score, and the penalty score. It also shows the number of priority assignments missed relative to the best solution at a specific priority level.

Algorithm	**Assigned**	**FOM**	**Penalty**	**# missed relative best at pri=**
Sequential	3608	0.5592	0.050	-32 at pri=16
Sequential with random profile internal FOM 50 tries	3636	0.5558	0.067	-7 at pri=16
Sequential with random profile external FOM 200 tries	**3644**	0.5645	0.062	**Best case primary criteria**
Greedy External FOM 30 tries	3584	**0.5775**	0.002	-2 at pri=6

Table 6:NASA Case Study Results

It turns out the best solution from a FOM point of view does not meet the primary goal: "assign higher priority tasks before lower priority if possible". The best FOM case of 0.5775 found with Greedy, results in 2 fewer tasks assigned at priority level = 6 than a case with a FOM= 0.5645. To get the better FOM, assignments were made to meet the target usage levels at the cost of total tasks because that's what the FOM said was the trade-off. A human operator would probably not bump a higher priority task in order or take a lower total number of assignments to obtain better resource usage.

Another point of comparison is time to solve. All algorithms took around 10 seconds per try to solve the problem except for the Greedy FOM which was significantly slower at over 300 seconds per try. Trying to

add code that could search for very small improvements in the external FOM was very expensive computationally. On the other hand method 3 which loosely coupled the external FOM but allowed for random searching of the solution space found the best solution from the primary goal point of view using less computation time.

Conclusion

In this paper we have shown how creating customized algorithms using STK Scheduler can be used to increase the value of schedule solutions. Optwise provides the underlying architecture with a flexible algorithm scripting language and solver engine. On top of the Optwise architecture, STK Scheduler provides GUIs and APIs that allow the user to define tasks and resources that can be tied to physical constraints, build specialized algorithms, adjust the figure-of-merit, run the algorithms, and visualize the schedule results in a variety of formats. The value of the custom algorithm was shown in a NASA case study where a variety of custom algorithms were developed and compared with regard to how well each solution met a variety of specific criteria. It is the design synergy between adaptable algorithms, schedule data management, flight dynamics computations and user interfaces which is required to meet the aerospace scheduling challenges of today and tomorrow.

References

[1] Barbulescu L., Watson J. Whitley L. and Howe A. *Scheduling Space-Ground Communications for the Air Force satellite Control Network* Journal of Scheduling, Vol. 7, 2004

[2] Fisher W. *Scheduling Algorithm Technology STK/Scheduler* 2004 AGI Users Conference, July 2, 2004

[3] Fisher W. *STK/Scheduler Next Generation Scheduling Algorithms Under the Hood Presentation* 2008 AGI Users Conference, Oct 9, 2008.

[4] Ziegler,I. and George D. *STK Scheduler Online Product Documentation.* http//orbitlogic.com/support/Scheduler/frame.htm

[5] Kennedy M. and Chua L., *Neural Networks for Nonlinear Programming,* IEEE Transactions on Circuits and Systems, Vol. 35, No.5, May 1988

[6] Fisher W., Fujimoto R., and Smithson R, *A Programmable Analog Neural Network Processor,* IEEE Transactions on Neural Networks, Vol. 2, No. 2, March 1991

[7] Fisher W. *The Optwise Corporation Scheduling Algorithms (As used in STK/Scheduler)* In STK Scheduler product documentation.

[8] Zweben M. Davis E. Daun B. and Deale M. *Scheduling and Rescheduling with Iterative Repair* IEEE Transactions on Systems, Man Cybernectics, Vol 23 No 6 1993.

[9] Chien S. , Knight R., Stechert A. , Sherwood R., and Rabdeau G. *Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling* Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling 2000.