Rev 7/15/2004

The Optwise Corporation Deconfliction Scheduler Algorithms (As used in STK/Scheduler)

Introduction

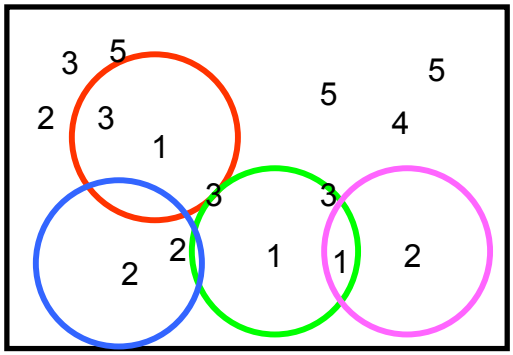
Scheduling is a complex process. In the case of STK® Scheduler this process has been broken into three divisions of labor matching the strengths of three companies. Combined, the strengths of these companies produce a single functional unit from a user perspective. Analytical Graphics Inc. though STK provides the world context or model from which all data used as the basis for scheduling is validated. STK also provides a means of visualizing the scheduling results in the world context. Orbit Logic provides the intuitive user interface and "traditional" Gantt, resource and task assignment listing views functionality. Most importantly, it coordinates the users scheduling related interaction with STK and the Optwise scheduling algorithms. This white paper is focused the final piece, the scheduler algorithms and framework provided by Optwise Corporation.

Before discussing the specific scheduling algorithms it is useful to understand some background concepts.

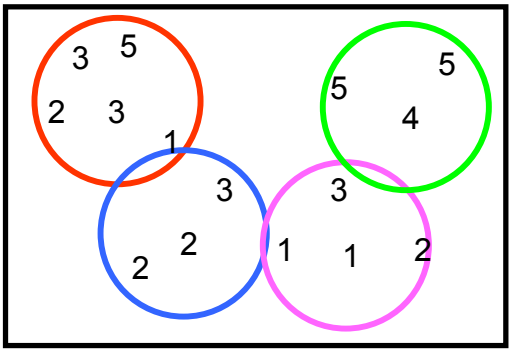
De-confliction and Optimization

De-confliction is the process of finding a solution that obeys all physical constraints. In the example below a sensor can be pointed at only one location and there are four sensors. Also, the sensors must point directly at one of the numbered locations and can only be pointed once. The numbers indicate the priority of the locations. A de-conflicted solution may or may not be optimal when measured by a figure of merit. An algorithm that assigns sensors in priority order produces a valid de-conflicted solution but only covers 7 of the 15 targets. Another algorithm could find a solution covering all of the targets.

Priority Driven



Coverage Optimal



Which is better? It depends on the criteria used to judge the final result. If it is critical that the highest priority locations have a sensor centered on them then the priority driven solution is optimal. If covering as many targets as possible as long as they are in the sensor Field of View (FOV) circle then the solution on the right is optimal. Optwise Corporation has a range of algorithms that do simple de-confliction, as in the priority driven example to advanced algorithms that find more globally optimized solutions.

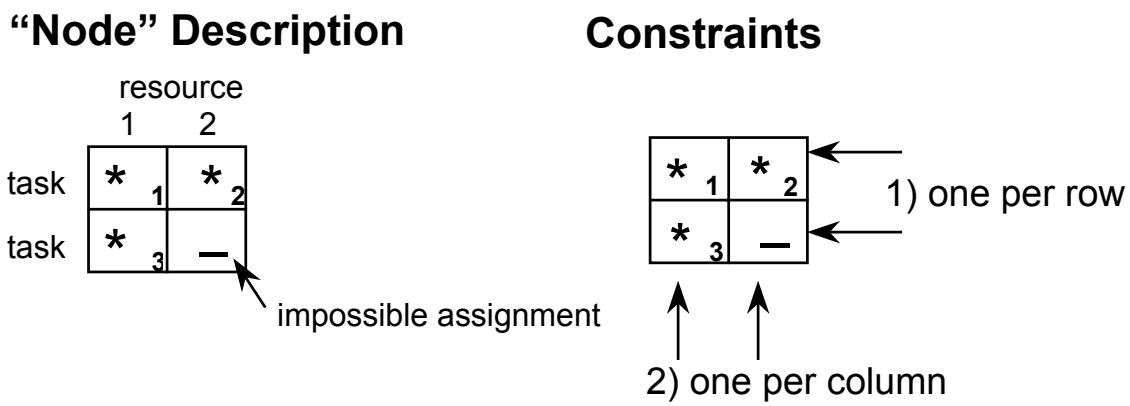
All of the algorithms discussed here are designed to produce conflict free solutions. If the algorithm cannot find a conflict free solution when an attempt to "assign" a task is made, then the task is left unassigned.

Rev 7/15/2004

STK/Scheduler Task-Resource Assignment Algorithms

To understand the different characteristics of the scheduling algorithms available in STK/Scheduler it is useful to consider a simple assignment problem below. The scheduling problem is just a more difficult version of the assignment problem.

Consider the two tasks and two resources problem in the figure below. Prior to running an assignment algorithm system modeling tells us which assignments are feasible. In the figure the stars represent physically possible assignments and the dashes impossible assignments. In this example task two cannot use resource two. The possible solution nodes are numbered 1-3. The concept that a task only requires one resource assignment for solution can be expressed as a rule "one per row" while the concept that a resource can only be used once can be expressed as a "one per column" constraint.

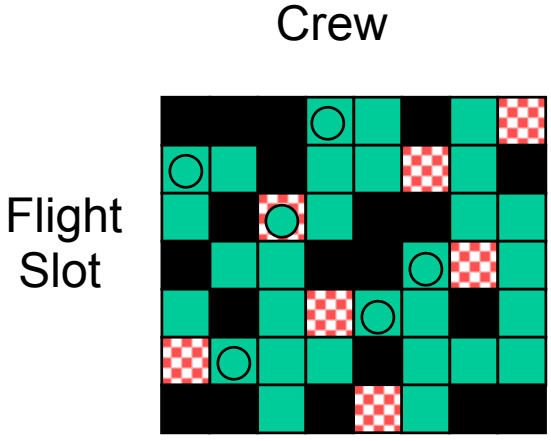


Sequential Algorithms

Sequential algorithms are algorithms that assign tasks one by one following a sequence based on pre-defined rules. First consider a sequential algorithm in which tasks are assigned in task numeric order and resources are used in resource numeric order. In such a case, the second task is blocked because node 1 blocks the use of node 3 (and node 3 blocks node 1). However, if task 1 is assigned node 2 then node 3 is available for use with task 2.

Real problems have many more tasks, many more possible resources (or combinations of resources) and more complex constraints. There are many versions of such sequential algorithms that are used to solve them. However, in general the strategy is similar: 1) choose an order to try tasks and possible resources, 2) test if a new combination is possible and keep it if is, or 3) use some method to find the cause of the block and remove it if possible.

A larger example of this type of assignment problem is shown at right, framed as the assignment of crews to flight slots. In the figure green (or gray) indicates that an assignment is feasible, black means the assignment is impossible and the crosshatch indicates that that combination has been assigned. In this example four crews are capable of satisfying the first slot, six the second and so on. There are 4*6*5*5*6*7*3 = 75,600 possible assignment combinations (most illegal). One of several legal and optimal (maximum assignment) solutions is shown using the checkerboard squares.



A typical sequential algorithm assigning the first

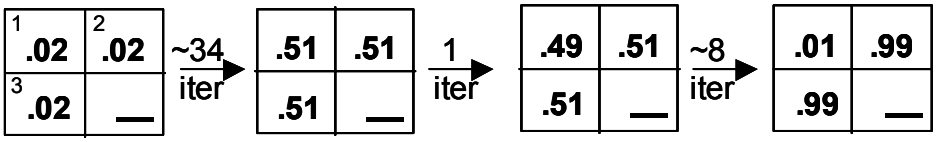
available (left to right) would not be able to assign the last flight slot as shown by the circles. On the other hand, backtracking to row five and changing its assignment to column 7 or 8 would allow the last flight slot to be assigned.

Most sequential algorithms work in this fashion. If the designer has made a good guess for the rules governing the task and or resource search order the solution can be quite fast. On the other hand a bad guess (or problem not suited to the strategy) most or all of the possible combinations may be tested as bad decisions are undone and new combinations are tried. For problems just a bit larger than this one, a search all of the combinations becomes unrealistic. With each additional task the number of combinations is multiplied by the number of possibilities for the additional row. For a 17 tasks, 17 resources problem the number of combinations is 3.6×10^{14}. Assuming 100 operations per combination a 2 GHz processor would take 208 days for an exhaustive search.

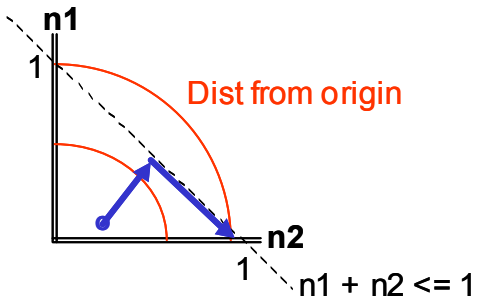
Thus most sequential algorithms balance the need to search all possible combinations with the need to finish in a reasonable amount of time. Most quit when a feasible solution that assigns all tasks is found. More sophisticated algorithms use the information in the constraints to make backtracking more efficient or limit the number passes by trying to discover the best candidates from prior knowledge of the domain.

A Non-Sequential (Global) Search Algorithm

Now consider another type of algorithm. This is not the only way to do global optimization, but is one that has been found to be particularly useful as a starting point for the Optwise de-confliction algorithms. Again consider the two tasks, two resources problem from earlier. Let us start by setting all of the nodes to a value near zero. We will then allow them to grow using the simple rule that the new node value is equal to the old times 1.1; a 10 percent growth rate. A node value will double in just over 8 iterations. A node starting at 0.02 would reach a value of 1.0 after about 41 iterations. Now imagine we place some constraints on this growth. First, nodes will be stopped when they reach a value of 1.0. Next, if the sum of the nodes on any row is greater than 1.0, every node in the row will have a penalty subtracted from its sum proportional to the amount the row sum was over the 1.0 threshold. We will do the same for the columns. The values of the nodes would follow a sequence similar to the figure below.



After about 34 iterations all nodes would be close to a value of 0.51. This would violate both a row 1 and column 1 constraint. Node 1 would be penalized twice and nodes 2 and 3 would be penalized once. If the penalty is half the amount of the constraint violation, then the new node values would be .48, .49, and .49. One more iteration would change the values to .49, .51, .51. Because nodes 2 and 3 now have a size advantage over node 1, subsequent iterations will increase the values of nodes 2 and 3 at the cost of node 1. Once nodes 2 and 3 reach 1.0 the hard node maximum constraint would stop any further changes in the "state" of the system. Consider the figure on the right showing the evolution of a problem for node 1 and node 2.The path in this "solution plane" is represented by the blue arrow. The node growth term pushes the sum of node 1 and 2 away from the origin until the constraint n1 + n2 < =1 is reached. The path then moves to the corner where n2 =1 and n1=0. Notice that the distance from the origin always increases. The dynamics of the underlying node growth term maximizes the distance from the origin. If the node and constraint representation are



chosen well, the maximization of distance will also correspond to maximizing the assignment of tasks to resources. Fortunately, this mathematical complexity is hidden from the user. The Optwise Interface does the required mapping behind the scene.

Although the example above goes to the optimal solution, it is not the case if node 1 is given an initial value significantly larger than nodes 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. The resulting stochastic algorithm will produce solutions that tend to maximize the number of assignments, but may be slightly different. For example, consider a one task, two resources problem. It can also be mapped to the last figure on the prior page. Node 1 represents the assignment of the task to resource 1 and node 2 represents the assignment of the task to resource 2. If the initial seeding is such that node 2 is larger than node 1 (as shown in the figure) than the solution will evolved to the node 2 =1 node 1 =0 solution. However if node 1 is initially larger than node 2 than node 1 will end up as the solution. The case of node 1 = node 2 must be broken with a tiebreaker. This problem is said to have a 50% probability of arriving at solution 1 and a 50% probability of finding solution 2. More complex problems cannot be analyzed so easily, and a great deal of research effort went into how to map “typical” problems into node representations that had a high probability of yielding maximum assignments. This algorithm is not guaranteed to find the maximal assignment but is a very efficient method having a high probability of finding one of the maximum assignment solutions. When evaluating the performance, sub-optimal solutions will occur. It is best to run the algorithm multiple times and choose the best solution.

Notice that unlike a sequential algorithm there is no task or resource order rules and that backtracking to good solutions is built into the method. The constraints provide the feedback. Further the problem solution cost grows with the number of nodes needed to represent it. Typically there are as many resources as tasks so the complexity of an assignment problem is on the order of the square of the number of tasks. In tests of the actual implementation the calculation cost scales worse than number of tasks squared because simple problems require less time to settle to a final solution (fewer iterations). For a particular set of problems design to stress this algorithm, the solution time scaled as $D^{4.2}$, where D is the number of tasks. The particular test represents the worse case. For most problems the scaling is closer to $D^{2.5}$. This is far better than a search of the combinations that increase as D factorial. While the scaling is much better, the setup cost for this global algorithm is greater than for a sequential search method. Thus for small problems this global algorithm will be slower.

In STK/Scheduler the corresponding scheduling algorithm is known as the Neural algorithm since neural network research inspired its development.

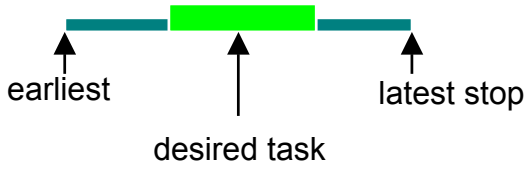
For those mathematically inclined the technique for mapping this type of assignment problem into an analog Neural network processor is described in Kennedy and Chua, Neural Networks for Nonlinear Programming, IEEE Transactions on Circuits and Systems, Vol. 35, No.5, May 1988, and in Fisher, Fujimoto, and Smithson, A Programmable Analog Neural Network Processor, IEEE Transactions on Neural Networks, Vol. 2, No. 2, March 1991.

Scheduling

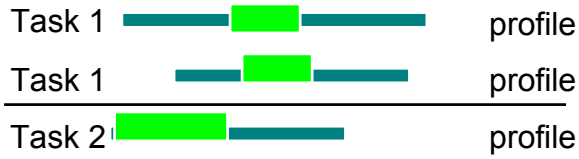
Schedule de-confliction requires that physical as well as temporal constraints be resolved. As will be seen in the series of demonstration examples, the Optwise de-confliction scheduler has evolved to have a variety of algorithms including one based on the Neural search method described above for solving scheduling problems. The algorithms fall into two major types, sequential and stochastic as introduced above. The specifics of the algorithms will be discussed in a series of examples after a brief discussion of the Optwise Scheduler Model.

As shown in the figures below, tasks are satisfied when a solution profile is assigned to them. These solution profiles are derived from task to physical resource access times (such as a satellite to target access) and may combined with other resource related properties (such as

satellite onboard memory). However, as far as the scheduling algorithms are concerned all problems start with time slots and the associated solution profile.



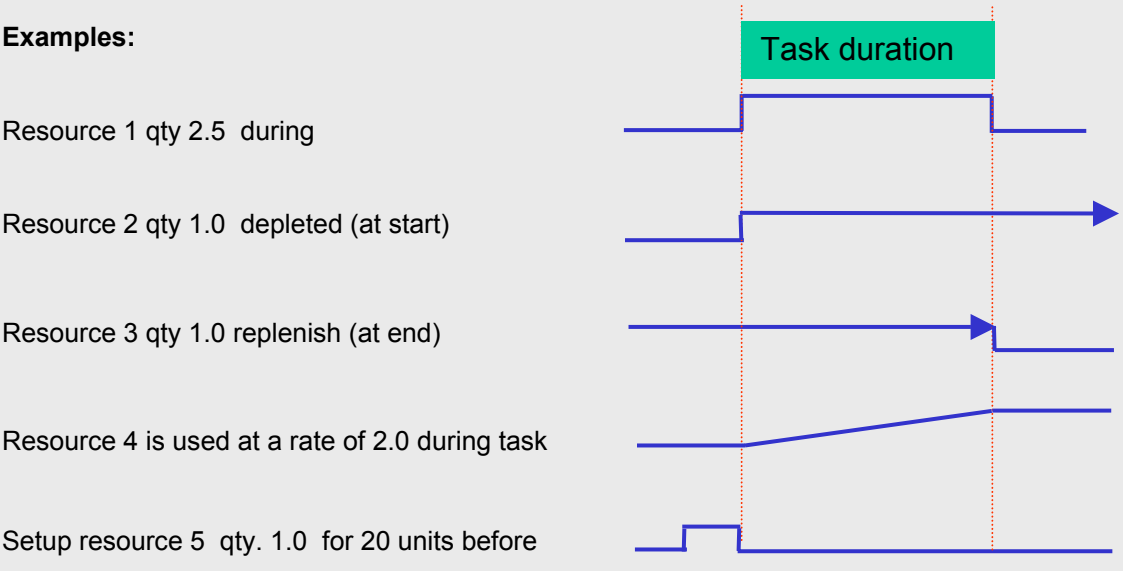
Tasks are scheduled onto time slots which represent the time during which a particular resource or combination of resources (a solution profile) may be used to satisfy the task.



Multiple time slots with the same or different profiles may be possible for a given task

Resources may be used during, allotted at the start, or replenished at the end of tasks. A setup resource may be defined for use prior to the timeslot access.

Examples:



The Optwise algorithm interface also allows each time slot to be given a floating-point value describing its desirability. This allows the algorithms to differentiate between otherwise equal time slots. The process of creating the feasible time slot data can be as simple (a point to point STK access calculation) or complex (multiple calls to STK calculate accesses with internal STK

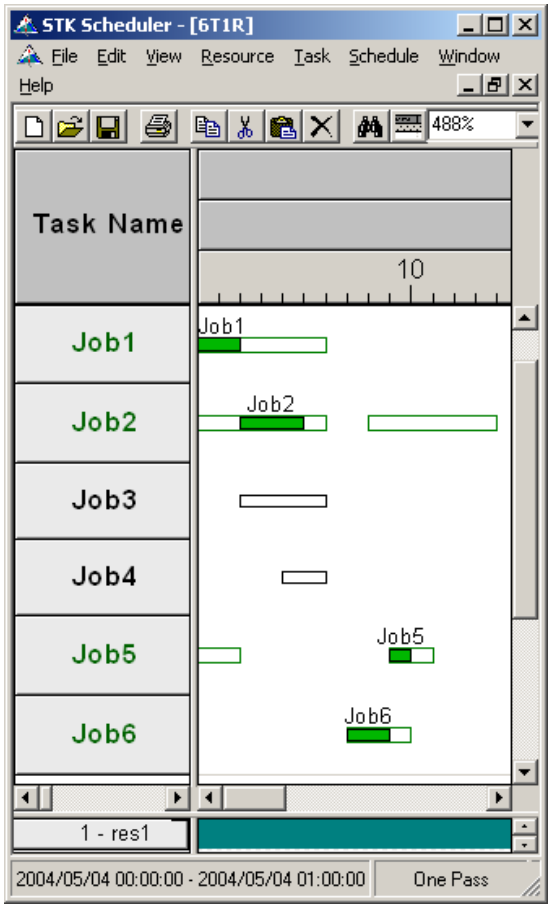
constraints combined with STK external user generated constraints). Periodic tasking for stereo imaging might be an example of a complex case.

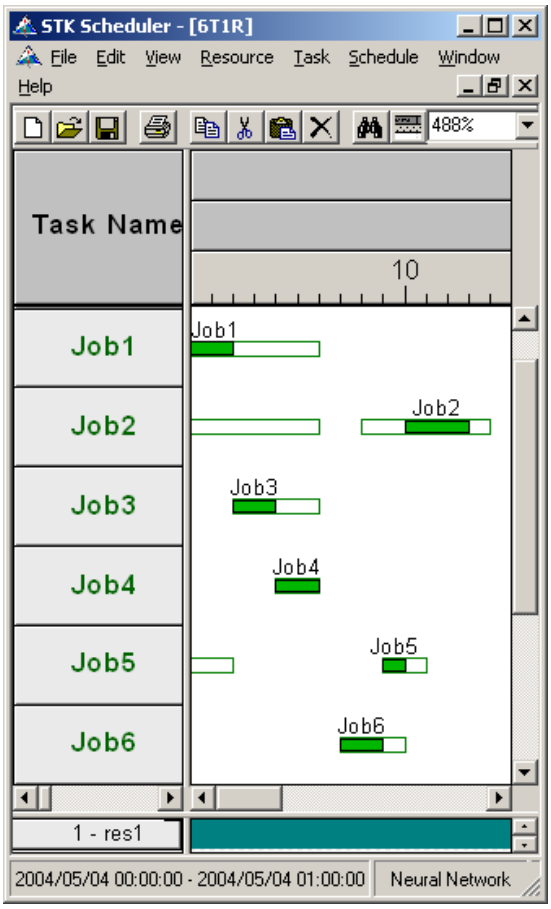
The job of each of the algorithms is to pick a time slot and a start time for the task within the time slot. Optwise assumes that even though there may be many possible time slots for a given task, only one will be used. There is an implied OR. AND conditions between resources are implied within the solution profiles. More complex Boolean combinations are handled prior to hand-over to the scheduling algorithm interface within the user interface of STK Scheduler.

An Example Tour of the STK/Scheduler Algorithms

In this next section an example or two will be used to introduce each of the algorithms illustrating how each algorithm might be matched to a particular problem.

In the example below six tasks of varied desired duration have one or more feasible time windows that can be used to find a solution. The feasible time windows are drawn with open rectangles. Tasks 1, 2, and 6 require duration 2 while jobs 2 and 6 require durations 3 and 1. There is only one resource that has capacity one. Scheduled task duration is indicated by a filled rectangle.

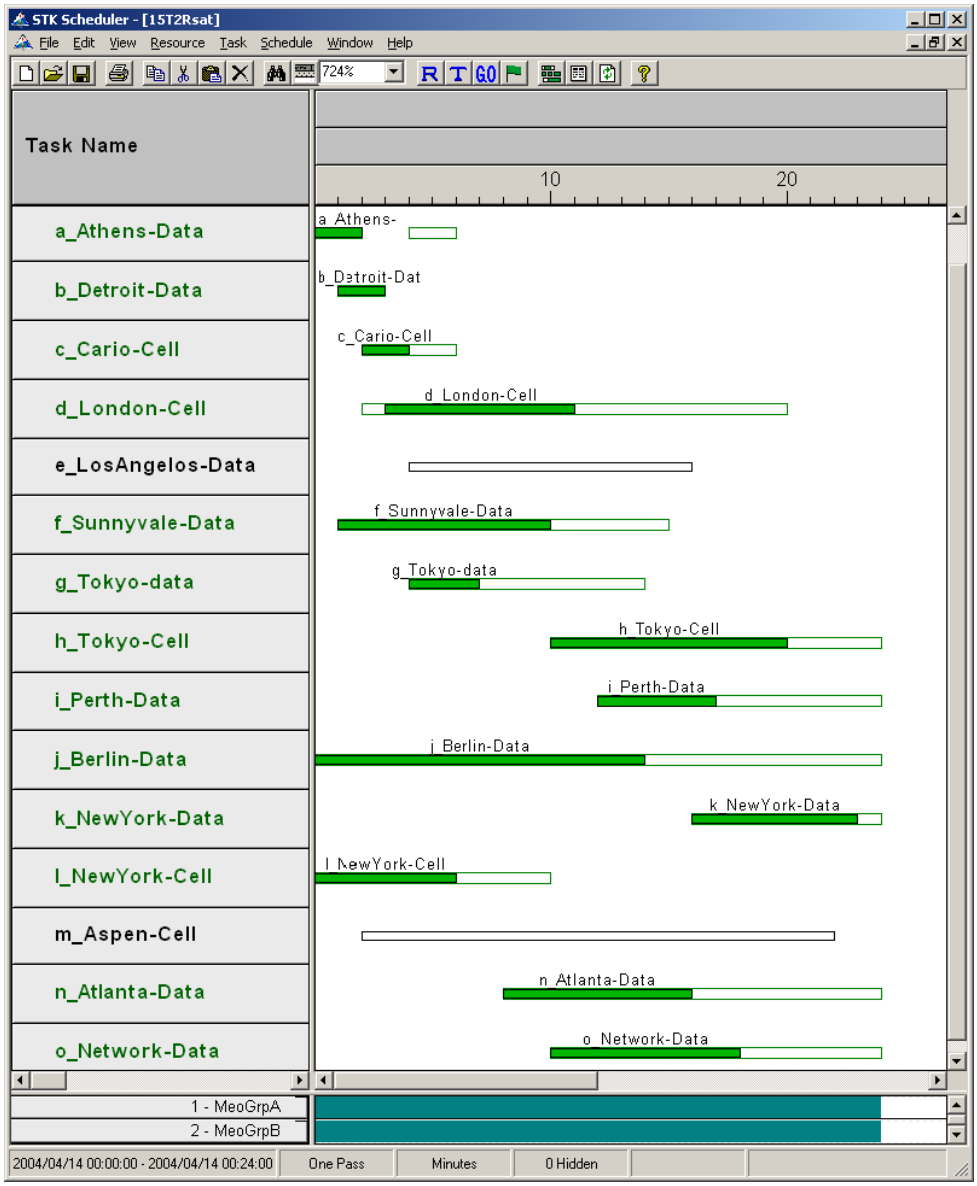




The figure to the left shows the solution found by a one-pass scheduler (OPS) algorithm that assigns a task to earliest possible time in task priority order. The task priority used to guide the consider order is the same as the list order. Task 3 and Task 4 cannot be assigned because the resource has been used by tasks 1 and 2. The figure to the right shows a full solution generated by the neural network search algorithm (Neural). The Neural scheduler uses an algorithm that evolved from the neural network method described in the assignment problem section. Initially all

of the possible time slots are seeded with a start time and duration appropriate for the corresponding task. The state of the system is iterated allowing the task assignments to be adjusted within the time slots. As the task positions are adjusted they interact with the resource limitations and implicit constraints such as "each task may have only one solution". Illegal and less favored assignments are driven to zero while the assignments that obey all constraints are driven to the assigned condition. The Neural algorithm finds the full assignment (all tasks assigned to a solution profile slot) solution 33% of the time, close to the theoretical maximum for this particular problem.

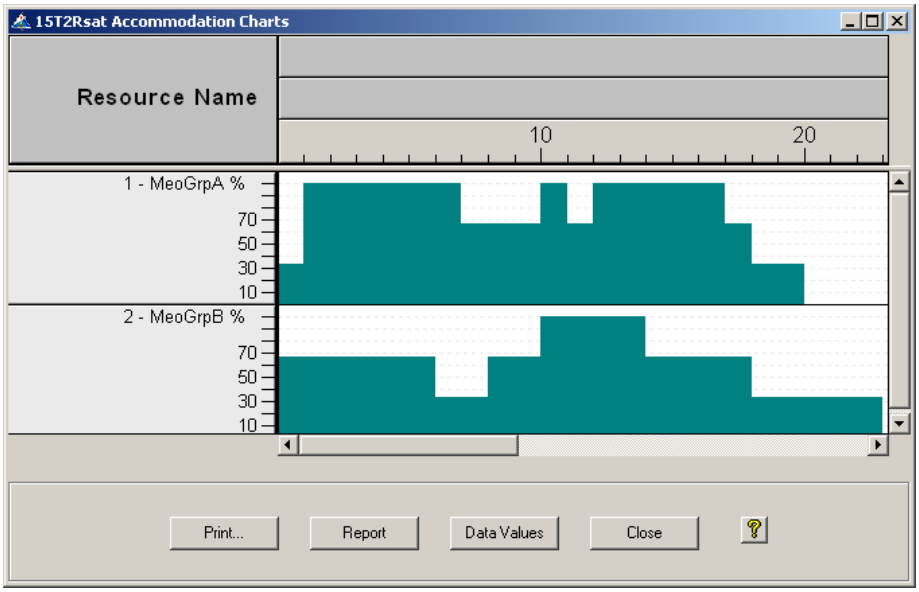
A more difficult 15 task, two resource problem is shown in the next figure along with the solution generated by the OPS algorithm.



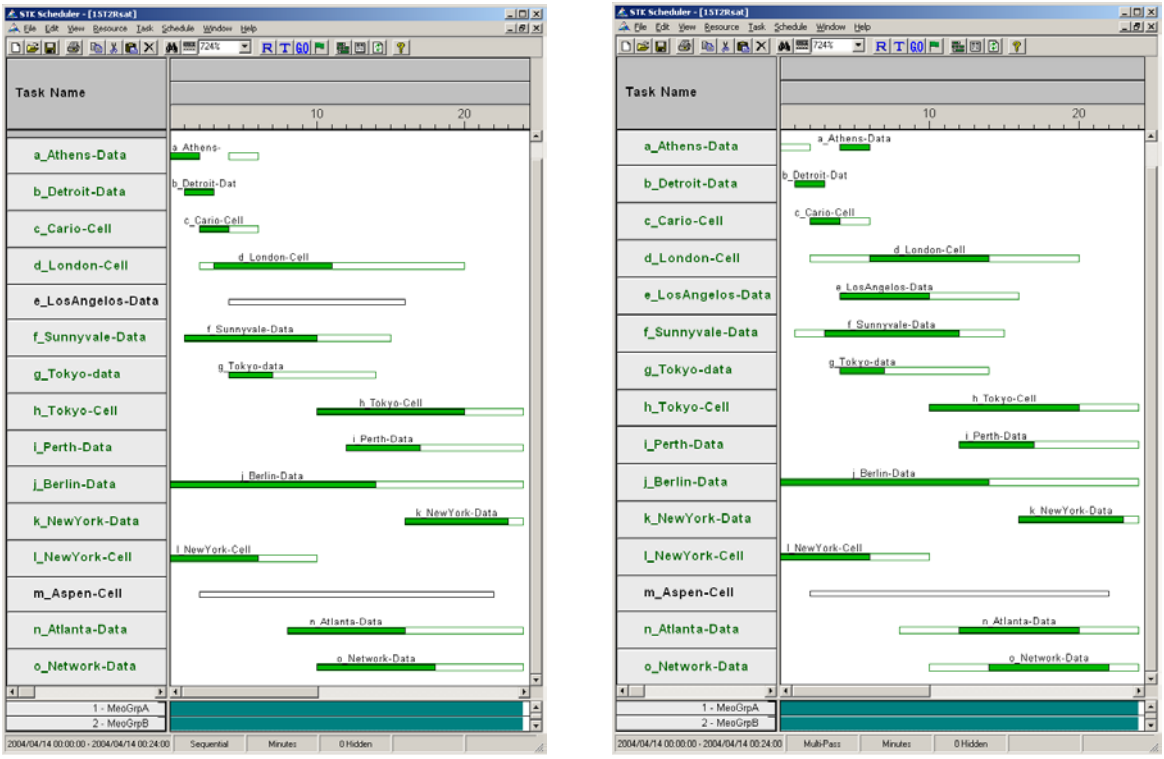
The two resources MEO Grp A and MEO Grp B each have a capacity of 3 (tasks). Some tasks can be satisfied with resource A, some with B and some with either. On the Gantt chart tasks that can be used by either resource show up twice on the task list when the list is expanded. There

are two possible task resource combinations of which only one should be chosen. The last task, network-data requires both resource A and B. This problem has far greater resource interaction than typical satellite problems. Note that the OPS algorithm finds a solution with 13 of the 15 tasks scheduled. The order in which the tasks were considered follows the task priority.

The corresponding resource usage for the15 task problem is shown below. Note while this problem has high resource usage, there is available resource time for the two missing tasks if the task placement was improved.

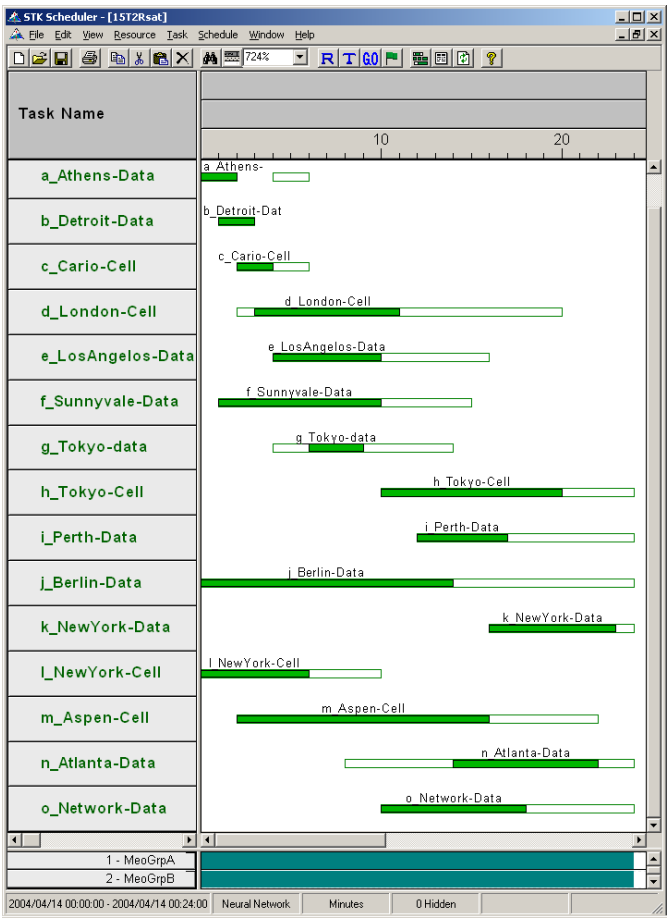


Two other scheduling algorithm options available are the Sequential algorithm (Seq) shown on



left and the Multi-pass scheduler (MPS) is shown on the right in the figure. The sequential

algorithm uses the time sequence of the available time slots. For all tasks with the same priority, time slots are tried in the order of the time slot start time. This algorithm is well suited to problems where there a just a few possible time slots and where the conflicts are weak. In this problem where there are many conflicts two tasks are missed. The MPS algorithm makes multiple passes using the OPS algorithm but modifies the task order and the resource order prior to each pass. A set of expert system rules based on Optwise Corporation past experience is used to generate the possible task and resource order lists. As can be seen the algorithm is able to schedule one additional task. Each pass is graded based on a figure of merit (FOM) and the best solution is returned. The figure of merit will be discussed later. For now assume it will find the best full assignment solutions.



The figure at left shows the result for a typical run of the Neural algorithm in which all of the tasks are scheduled. The neural algorithm is able to schedule all tasks because it uses the resource interaction information to find the solution.

As was mentioned earlier, if a series of runs are made, the Neural algorithm does not find a single solution but will find a family of solutions that tend to maximize the assignment (or scheduling) of tasks. Because the Neural algorithm is stochastic it is possible to specify a number of trials to run the algorithm from which the run with best result is used. (The “best of” on the Schedule/Properties menu.) On any given run this stochastic algorithm has a probability of finding a particular solution. For this problem the probability of finding a full solution (15 tasks assigned) is about 70%.

Another method of searching for a solution in an un-bias manner is to seed a random solution (an available option) and using a Repair algorithm to repair it. The Random-Repair algorithm is a particularly good algorithm to use when there are many identical resource which need to be used uniformly. Random can also be run in a “best of” mode. For the 15 task problem the probability of finding a full solution is about 50%.

So in the examples we have introduced the One pass (OPS), Sequential (Seq), multi-pass (MPS), Neural (Neural) and Random with Repair (Random). Which is better? That brings us to the subject of Figure of Merit or FOM.

Figure of Merit

Internally, each of the algorithms finds solutions using different methods. The sequential algorithms OPS, SEQ, and MPS have at their core a set of rules that are followed to place each new task on the schedule. Within the Neural algorithm there is a simple goal, schedule the greatest amount of task time as possible as early as possible. Rand has an internal goal to

generate unbiased schedules. This general goal is modified by a very loose coupling to external figure of merit (FOM).

After any of the algorithms completes the resulting schedule is graded with an external figure of merit. Currently, the following user programmable FOM is available in STK Scheduler.

$$FOM = \sum_{task=i} weight_i \left(\begin{array}{l} K_{assign} * \left(\frac{duration_i}{desire_i} \right) + K_{dur} * duration_i \\ + K_{PP} * Possibility\,\mathrm{Pr}\,iority_i + K_{PN} * PositionRanking_i \\ + K_{early} * EarlyBonus_i + K_{\max Dur} * MaxBonus_i \end{array} \right)$$

where

$$weight_i = \frac{l\arg est\,\mathrm{Pr}\,iority - smallest\,\mathrm{Pr}\,iority + 1}{Task_i\,\mathrm{Pr}\,iority}$$

and

$$EarlyBonus_i = \frac{TaskLastestStart - ActualStart}{TaskLastestStart - TaskEarliestStart}$$, (1 if latest start = earliest start)

The FOM has five user definable constants (the "Ks").

K_{assign} is used to adjust the relative weight for pure assignment since duration/desired(duration) is one for a full 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_{PP} is used to adjust the relative weight of Possibility Priority which is average resource priority for the timeslot.

K_{PN} is used to adjust the relative weight of Position Ranking of the task timeslot. Timeslots that are closer to the scheduling preference will have a higher ranking. See STK/Scheduler help for details on how poison ranking is calculated.

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

K_{MaxDur} is used to adjust the relative weight of the Max duration bonus. By time of release of STK Scheduler an additional function will be available that grows all task durations from the minimum (desired) duration to a maximum duration. This term will reward additional duration in a linear fashion similar to the early bonus term.

The desirability of a particular time slot is the sum of the K_{PP} and K_{PN} terms. Setting K_{assign}, K_{PP}, K_{PN}, K_{maxDur} = 0 and $K_{duration}$, K_{early} = 1 will give the default FOM used for the remainder of the

white paper . This default figure of merit weakly awards tasks that are scheduled earlier and will give more credit to tasks with longer durations if the priorities are the same.

The table below shows the FOM calculated for each of the prior runs of the scheduler algorithms. The assignment success is also noted. For the Random and Neural algorithms the mean and standard deviation is given for 100 runs.

Algorithm	**OPS**	**MPS**	**Seq**	**Random**	**Neural**
# assigned, best	13	14	13	15	15
Prob. of 15/15	0	0	0	48 %	72%
FOM, best (100)	11,549.8	12,030.3	11,549.8	12,869.7	12,870.2
FOM, ave. (100)	-	-	-	12,304.8	12,646.0
FOM, stdev.	-	-	-	647	368

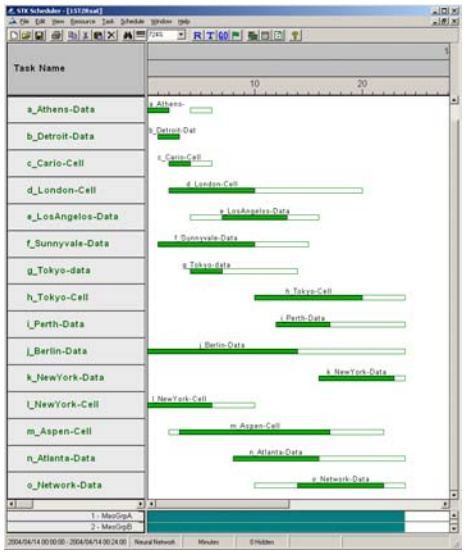
Of the algorithms, MPS which is really multiple runs of OPS, is most strongly coupled to the FOM above. The solution chosen is always the run with the best FOM. On the other hand, for the Neural algorithm, the user must make multiple runs and choose the best result. The initial desirability data and FOM constraints weakly affects the yield from this algorithm.

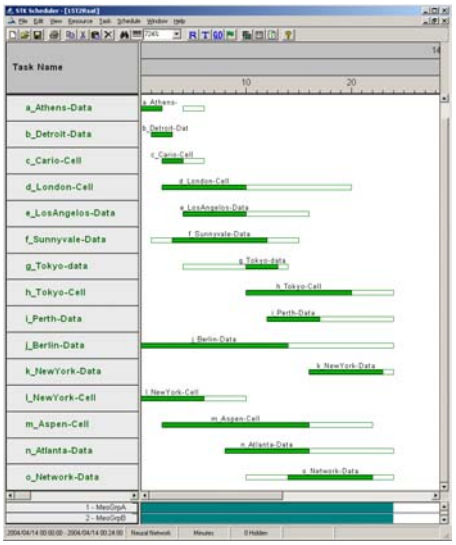
OPS is primarily driven by the priority of the tasks. However, its search order is also biased by the timeslot desirability and the resource list order. The search order can be affected by changing the FOM constants as well. For example, if the constant controlling desirability is set to zero than the desirability presort is turned off.

At the other extreme using Random algorithm completely ignores the FOM on an individual run. It is only used as an external critic. This combination is useful when a completely unbiased search is needed. This might be the case if the user is trying to optimize for something that can not be programmed in the current FOM.

The fact that the FOM calculation is not deeply embedded in the algorithm, allows for some flexibility in creating enhancements to the FOM to adapt it to external variables that are hard to capture in resource constraints. If a family of solutions is generated using the Neural or Random algorithms, any external critic including a schedule review board can make a decision on the final best schedule.

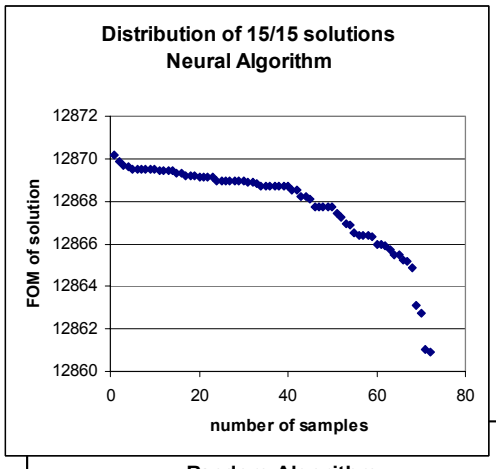
Two solutions from the family of solutions are shown in the next two figures. The FOMs are 12,870 and 12,865.

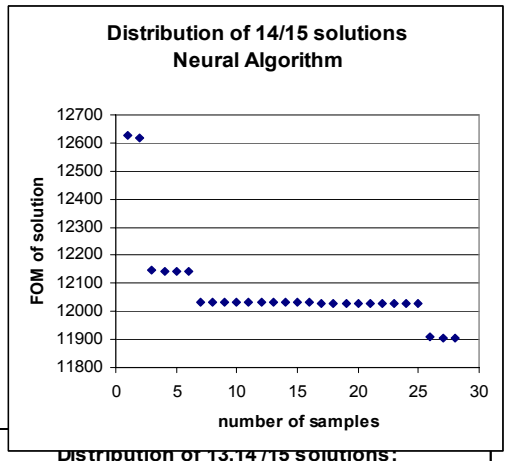




Distribution of Global Scheduler Solutions

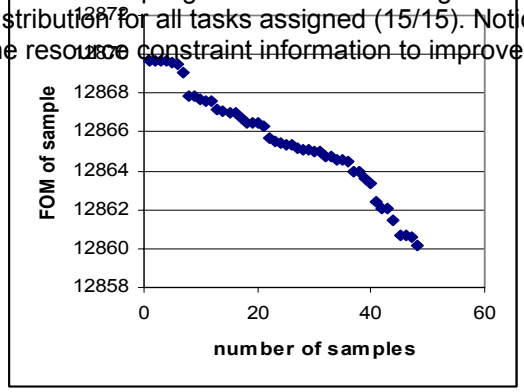
To illustrate just how large the family of "good solutions" is for even a highly constrained problem such as 15T2rSat a series of 100 runs were made with the Neural algorithm. The full assignment (15 of 15 tasks) solution was found in 72 of the cases (72% yield) with the distribution of FOM shown below. Each point corresponds to a unique solution differentiated only by the number scheduled and the FOM. Thus within the criteria that all tasks are assigned it is possible to differentiate the quality of the solutions even further. In this case on the K_{early} term is making the difference. Some tasks start earlier in their time slots than others. The 28 trials that resulted in only 14 scheduled tasks had the distribution on the left. The larger variation in the 14 of 15 solutions is because a task durations vary from 120 to 840 and weights from 4 to 1 and the task weight times task duration term dominates the FOM. The large jumps occur when different tasks are not assigned.

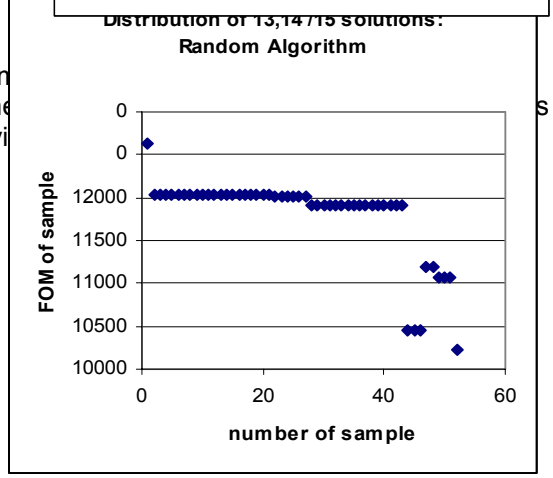




Random Algorithm

A similar sampling can be created using the random
distribution for all tasks assigned (15/15). Notice the
the resource constraint information to improve its yi





The left plot shows the distribution of 13 and 14 of 15 assignments with the 14 assigned first followed by the 13 assigned solutions .The FOM is then sorted within each group. The increase in FOM in the 13, 14 assigned task distribution at sample 45 is due to a trade off where assigning two tasks of lower priority gives a lower FOM than assigning one higher priority task. For this set of FOM coefficients assigning fewer tasks is better.

While the neural algorithm is clearly better in this case, every problem will have a different distribution and the distribution will change as the coefficients of the FOM are modified. The best way to know is to test.

Performance with Problem Scale and Type

Each of the algorithms will have types problems where they are particularly well suited.

The One Pass algorithm will in general be the fastest because it is the simplest internally. To first order the one pass solution time scales with the number of tasks if the first available resource and access time works. However if there are many overlapping time windows, the solution time may scale as poorly as the number of tasks times the number of resource accesses per task. In general, problems that have many more resources than needed are perfect for the one pass algorithm. It is surprising how many problems (or sub-sections of a larger problem) fit this description.

Multi-Pass has similar characteristics to one-pass because it uses the one-pass algorithm internally. The number of passes it tries depends on the problem complexity. It does at least two and up to 30 depending on the number of resources available. MPS will always find at least as good a solution as one pass and can improve performance significantly for problems with moderate conflicts.

The Sequential Algorithm was developed specifically for scheduling problems where using the earliest contact is known to give good solutions. Empirically it has been found to produce very good solutions for problems dominated by ground access constraints. It is very fast but because it has a very structured search can perform poorly if the problem is not well suited to it such as problems with highly conflicted resources.

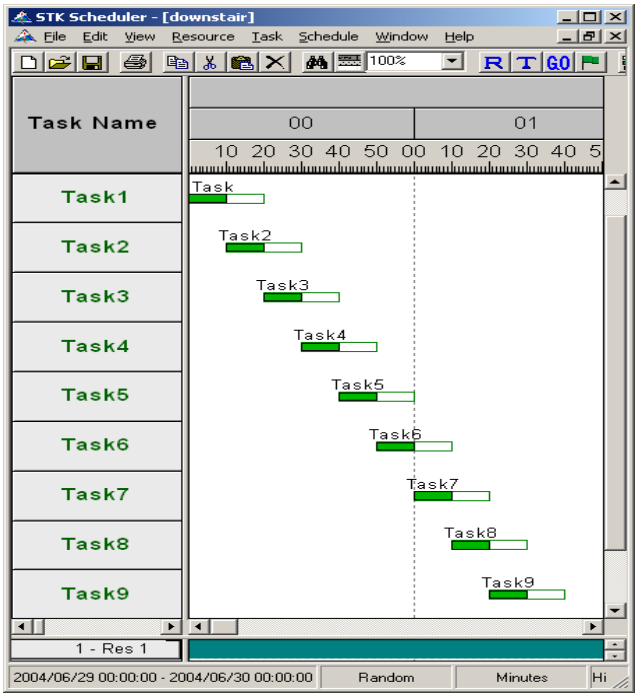
The Neural algorithm was designed to do highly conflicted problems. Because it searches all possibilities in parallel the time to solve will scale with the number of tasks times the number of resources accesses (the number of time-slots). Some simple problems will solve faster because the algorithm terminates itself when a stable full solution is found. The setup time is also higher for this algorithm. In addition to the single run computation costs, multiple runs will generally be desired to find the best n of m runs of the Neural algorithm.

The Random algorithm is another very fast algorithm with a solution time that is roughly linear with the number of tasks. It has the unique feature that it makes no assumptions about the structure of the problem so it can be very useful for sampling the range of possible solutions. It has proved very useful as the algorithm to use when it is necessary to uniformly load a set of identical resources. Normally Random is run multiple times using the “best of” option.

# Tasks	OPS	MPS	SEQ	Random	Neural
500	1	5	1	1	1
1000	2	39	2	2	4
1500	5	123	6	6	11
2000	8	201	9	9	18

The table at left shows the result of a scaling test run on each of algorithms for a problem in which each task had a one-access window twice as large as the desired duration. Each window overlapped the next task's window by 50%. The solution for the first 9 tasks and the available access is shown on the next page. It was designed so the required task duration just fit within the

available resource time. A different resource and profile was used for every group of 20 tasks. All times are in seconds and the testing was done on a 2.6 GHz Pentium 4 processor. Times do not include the STK/Scheduler display time which was less than 2 seconds. All algorithms were able to get 100% assignment. Because all tasks must fit perfectly and tasks must start as early as possible this is an easy problem for the OPS algorithm and Sequential algorithms find solutions immediately and have the best solution times. The other algorithms find the solution as well but at a higher computational cost. MPS is the worse computationally because it makes approximately 30 sub tries attempting to find a better solution. Neural has a larger setup cost in order to incorporate the resource constraint information.



Next, consider a satellite related scaling example. To create a 40 task set 10 targets were matched to four task types that had varying fixed duration times. STK accesses were obtained one day assuming simultaneous access to a ground station and some elevation constraints. Corresponding 80 and 120 tasks sets were created by matching 20 and 30 targets to the task types. Since all these example sets turned out to be over-resourced the 120 task set was rebuilt using a 10 hour time period. This reduced the total resource availability to a point where the scheduling algorithms needed to do some real work. The following assignment and FOM results were obtained. The FOM is smaller because the Kdur term was divided by 60 to have a FOM scaled to minutes rather than the default seconds. This problem is an example of one that is solved well by sequential. Only the Neural algorithm was capable of getting a full solution prior to the invention of the Sequential algorithm.

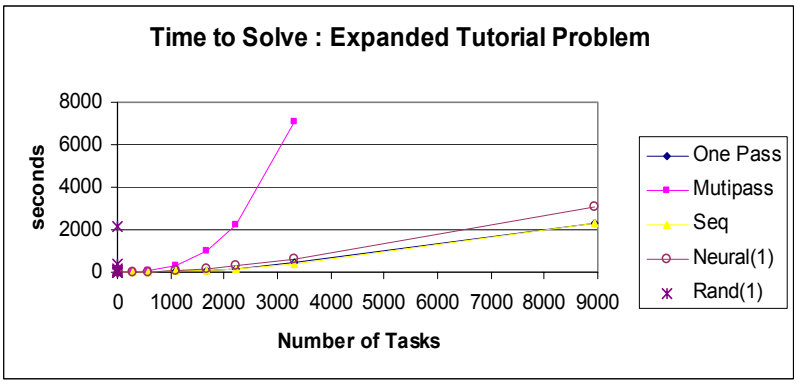
Tasks	period	OPS	MPS	SEQ	Neural	Rand
40	1 day	40, 79.1	40, 81.9	40, 81.9	40, 74.7	40, 68.4
80	1 day	80, 156.9	80, 162.4	80, 163.5	80, 129.6	80, 135.9
120	1 day	120, 232.6	120, 239.3	120, 232.6	120, 189.9	120, 206.0
120	10 hrs	116, 195.6	116, 217.5	**120, 238.8**	**120, 194.5**	116, 180.5

The corresponding solution times were all 4 sec or less and were dominated by I/O.

Finally, consider this scaling result based on a series of increasing schedule duration problems were created using the tutorial example provided with STK/Scheduler.

Schedule Duration	tasks	timeslots	One Pass	Mutipass	Seq	Neural(1)	Neural(5)	Rand(1)	Rand(5)
2	39	118	3	4	3	4	4	3	3
15	279	1141	12	20	12	14	25	12	13
31	574	2261	24	67	23	31	67	24	31
59	1090	4505	58	319	54	82	202	54	92
90	1662	6678	112	996	109	171	527	112	227
120	2215	9065	191	2221	182	289	999	179	455
180	3322	13571	432	7104	422	645	2529	405	1304
366	8953	27720	2293	57089	2271	3088	14168	2117	9141
days			sec	sec	sec	sec	sec	sec	sec

Since most of the tasks are periodic, increasing the length of the schedule naturally increases the number of tasks. The schedule durations varied from 2 days to 366 days as shown in the table. The final example had 8953 tasks with 27,720 possible access generated time slots. The tutorial example is mostly dominated by these access times but does require resource shuffling to obtain the best results. Typical assignment percentages were: OPS, MPS and SEQ 96%, Neural and Random > 98%. Since not all of the tasks could be assigned more than one time slot had to be tested for each task. Three or four time slots were available for each task. Now the OPS,SEQ and Random algorithms have less of a computational advantage over the neural algorithm. The Random and Neural algorithms were run once with “best of” set to 1 (Neural(1)) and once with “best of” set to 5 (Neural(5)). Notice that multi-pass is showing signs of non-linear scaling. Careful examination of the table data indicates that the other algorithms are not quite linear as evidenced by a declining task assigned per second rate as the number of tasks increase.



Special Conditions Affecting all Algorithms

All of the examples thus far have assumed that tasks use "resilient" resources, or resources that are used during a task. While this type of resource might numerically dominate a schedule just a few of special resource types available in STK/Scheduler can greatly influence the yield of a schedule. With the release of version 3 of STK/Scheduler (version 6.0 of STK) a number of enhancements were made that allowed users to define constraint relationships between tasks and to allow resource capacity to be rate based. A complete discussion of all of the complex interactions affecting the algorithms that this has introduced is beyond the scope of this white paper (perhaps a graduate student is looking for a good thesis topic). However, all of the algorithms are compatible with the additional features. Here are a few hints as to how the new features may affect performance.

Capacity Resources

Capacity resources are resources that are depleted (or consumed) or replenished either discretely or at a rate. At present a discrete depletion occurs at the beginning of task and discrete replenish occurs at the end of the task. Rate capacities are broken into a number of discrete steps spaced through the task duration based on a user definable resolution factor. A 5% resolution (the default setting) will break the capacity into 20 discrete level changes. The first level change occurs at task start if it is a deplete rate and the last level change occurs at task stop if it is a rate allocate. Clearly fine resolution may allow tasks to be scheduled closer but will add significant processing time for all cases. Use fine resolution only when needed.

Whether discrete or rate the value of the capacity at the end of the task persists after the task. This has huge implications for all tasking both before and after tasking. Consider a case in which one task depletes a resource (by one unit) and another task replenishes a capacity (by one unit). Clearly, the allocate task must occur earlier in time on the schedule than the deplete task (assuming the capacity resource begins at zero). But what happens if the replenish task has a lower priority (or is lower in the task list at the same priority) than the deplete task. For most of the algorithms which use some priority sorting to decide task ordering, only the “allocate” task would get on. The STK/Scheduler algorithms (including one pass) get around this problem by adaptively adding additional passes though the task list when such cases occur. Although this adaptive part of the algorithm is fast, it does add overhead to the scheduling process when needed. If it is known that a replenish task is only constrained by access (time slot) availability it will be faster to define the task with a high priority so it will tend to be scheduled first.

Soft Upper Limit Capacity Resources
A resource’s capacity can be defined to have a soft upper limit which will allow a task that replenishes the resource (either discretely or as a rate) to be on even if the resource reached maximum. For example, you might show a battery charge task as on every time there is access to the sun. When the battery reaches its maximum charge capacity the charging task will remain on but have no effect on the level. This can be useful when there are lower priory tasks that need to fit in around higher priority tasks. This unseen excess charge capacity becomes real capacity as tasking is added. This removes the need to more tasks to provide more charging. Using soft upper constraints can speed up processing and make a schedule easier to understand.

Task Precedence
When a task is defined as being a predecessor to another task the absence of the predecessor task can block the assignment of the task**.** If the blocking relationship is “n of m” then the task must obey n of m of the appropriate min/max time after task start/stop constraint between the predecessor task and the task if the predecessor is present. Placing task precedence constraints also causes the algorithm manager to adapt more passes as required to maximize scheduling with an increase in the solution time. Thus, use these only as necessary. Also see if it is possible to use a periodic task with a min/max time between re-occurring tasks which are more efficient.

Min/Max Time Between Re-occurring Tasks
As the name implies this is special type of constraint between known re-occurring tasks. Because the tasks are known to be re-occurring the algorithms can make short cut assumptions allowing faster scheduling. The definition also reduces the number of tasks that the user must specify.

Maximize no Handovers: Tasks that are marked as maximize no handover cause the interface layer of the software to run an expand algorithm after one of the base algorithms has been run. The base algorithm will have assigned (if possible) the minimum duration requested for each of the tasks in the schedule. The Expand algorithm expands the duration of each task out to the maximum time allotting in 5% increments starting with the highest priority task. Time or resource constrained tasks can be pushed later by earlier tasks that grow. This step can add significant processing time if there are a lot of time dependant tasks.

Maximize with Handovers
Tasks marked in this fashion ignored the algorithm selection (for just that task), using a special "handover" algorithm instead. The handover algorithm begins with the first available time slot and assigns as much time as possible from that time slot. It then looks for the next available time slot and repeats the process until maximum duration is obtained or there are no more possible time slots. There is no option to run a Handover algorithm because it is done automatically if the option is selected for the task. This type of task allocates as much of the available resource(s) in one pass (unlike expand) and one handover task using a critical task high in priority can easily lock out all other tasking. It can be relatively fast, particularly if the resources do not have a lot of interaction.

Summary of the Algorithms
The listing below reflects in summary form the characteristics of each of the algorithms and may be used as a rough guide as to which algorithm might be best suited to a problem. However, nothing is better than doing some testing for a particular class of problem.

One-Pass – Problems with many possible ways to do a problem without a lot of resource conflict. Algorithm is balanced to take into account task priority and time slot desirability. Time to solve scales linearity with number of possible time slots but can be even faster if many feasible solutions exist for each task

Sequential – Problems in which getting the earliest possible “on” time is critical. The earliest possible time slot is used within groups of tasks with identical priority. The time to solve is similar to One-Pass. A more descriptive name for this algorithm would have "Earliest time slot within a priority group", but that was too long so the shorter name Sequential has been used.

Multipass – Problems in which there is moderate resource interaction and where optimizing a FOM is important. Multipass is typically 30 times slower than One-Pass or Sequential but large variations possible due to problem complexity.

Neural – Problems with complex resource conflicts. Algorithm is balanced to take into account task priority and time slot desirability. The worse case time to solve scales roughly with the square of the number of possible time slots but typical problems scale very close to linear with the number of tasks. Has a multiple run option used to find best FOM from a family of feasible solutions.

Random – Algorithm that is used to optimize a FOM when an un-bias search is needed. An example might be trying to equally load a set of identical resources. The time to solve scaling is similar to One-Pass.

Concluding Remarks
The current STK Scheduler algorithms have been designed to solve a broad range of scheduling problems. However, in order to meet future needs the algorithms have been designed to be extended. AGI, Orbit Logic Inc. and Optwise Corporation look forward to our customer comments as we not only meet but also anticipate future scheduler algorithm needs.

About the Author and Optwise Corporation

Dr. William Fisher is the president of Optwise Corporation which he founded in January of 2001 to create commercial software components for optimization and scheduling.

Dr. Fisher began his connection with aerospace in 1987 when he joined Lockheed's Palo Alto Research Laboratory where his interests included precision optical pointing and tracking, the application of Neural network methods in the fields of adaptive control and high-speed computation and the investigation of non-linear dynamics in analog hardware systems. In 1995 he co-founded Applied Analytics Corporation, later renamed Anava Corporation where he served as the Vice President. At Anava Corporation he was the principle investigator on contracts related to the development of software and analog hardware for highly efficient optimization and scheduling algorithms.

In January 2001 Dr. Fisher founded Optwise Corporation as a friendly spin-off from Anava Corporation to create commercial software products based on the prior consulting experience.

William A. Fisher received a B.S. degree in Engineering Physics from the Ohio State University in 1977 and Master of Science and Doctor of Science degrees in Nuclear Engineering from the Massachusetts Institute of Technology in 1980 and 1983. During 1983-1987 he jointly worked as the Technical Director at National Nuclear Corporation and as a Visiting Scholar in the Physics Department of Stanford University.