Task Swapping: Making Space in Schedules for Space

Laurence A. Kramer and Stephen F. Smith
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{lkramer,sfs}@cs.cmu.edu

Abstract. Given their inherent expense, space missions must utilize available resources efficiently. Allocating resources to a single mission or scientific instrument is in itself a challenging task, and this complexity becomes magnified as fleets of satellites and networks of groundstations and antennas are managed as resource pools. Techniques which perform well for generating schedules for unit capacity resources, though, may not be as well suited for multi-capacity and multi-resource domains. Similarly, techniques which perform well at producing good schedules may not be appropriate for schedule improvement and schedule repair. While there has been increasing interest paid to multi-capacity and multi-resource domains in space, most existing approaches do not adequately address the problem of schedule stability in the context of schedule change. We present a general "task-swapping" procedure which is designed to improve oversubscribed schedules in an anytime fashion in multi-capacity and multi-resource space domains.

1 Introduction

Space mission planning and scheduling is an endeavor where there are rarely enough resources to support all of a mission's goals. There are always more deserving science observations that could be scheduled on the Hubble Space Telescope than actually can be scheduled. There is quite often more data that could be generated by the Landsat 7 than can be accommodated in the solid state recorder (SSR). Due in part to the expense of the resources involved and the complexity of managing them, automated planning and scheduling tools have been successfully employed to improve resource usage – making sure the highest priority tasks are allocated to provide the highest signal to noise ratio. In general, though, there is room for improvement, both in generating more highly optimized schedules, and in repairing schedules in response to unplanned events.

In this paper we review some of the approaches taken to maximize resource usage in space missions, and question whether they are applicable to addressing problems where there are multi-capacity resources and/or resource pools, where the schedule may fail, or where there is a need to incorporate additional tasks. Scheduling observations on an earth orbiting satellite (EOS) is a unit-capacity problem in that one observation can be assigned to a given science instrument at any given time. Scheduling the same observations to the solid state recorder (SSR) on that satellite is a multi-capacity problem, since more than one observation can share the data capacity of the SSR. Scheduling an observation on a fleet of EOS's is a multi-resource problem that takes advantage of resource pools that share the same characteristics.

We claim that many existing techniques for schedule generation and schedule repair fall short in addressing the increasingly important problem class of multi-capacity, multi-resource, oversubscribed domains. We discuss a task-swapping procedure (referred to as **MissionSwap**) (Kramer and Smith 2003; 2004) that has been successfully applied to aircraft mission scheduling, a domain that shares many attributes with space mission resource allocation problems. We generalize this procedure such that, given an appropriate task flexibility heuristic, it can address several common space mission scheduling problems.

2 Techniques for Optimized Schedule Generation and Schedule Repair

In the domain of space mission planning and scheduling a number of approaches have been applied to oversubscribed problems. Most of these rely on constraint directed search and can be characterized as either *constructive* or *repair-based*. Constructive techniques aim at building a schedule from scratch; they often rely on lookahead heuristics to identify potential resource conflicts and backtrack when confronted with an impasse. Repair based techniques progressively refine a given schedule state to address problems of resource allocation and infeasibility.

One early approach to augmenting an existing schedule with additional tasks was applied to a remote sensing satellite (SPOT). (Verfaillie and Schiex 1994) describe an algorithm, *local-changes*, that inserts a task into a schedule by unscheduling enough tasks to clear a space, and then rescheduling them, proceeding recursively on failure. The approach presented has the desirable properties of being applicable in any anytime fashion, and of attempting to preserve the stability of the existing schedule. This method

is very similar in spirit to our **MissionSwap** procedure, although it is confined to operate on a unit-capacity, single-resource problem.

A number of other efforts in the domain of space mission planning and scheduling have attacked the problem of oversubscribed resources, for instance, (Minton *et al.* 1992; Johnston and Miller 1994). However these approaches, too, focused on unit-capacity/single-resource domains. ASPEN (Rabideau *et al.* 1999) does tackle resource de-confliction across all resources on an Earth Orbiting Satellite (EOS). Although as (Frank *et al.* 2001) point out, the scheduling problem that it is applied to is very small, involving up to four observations a day.

Over the past several years, however, there has been increasing interest in addressing scheduling problems for space missions where pools of resources are available, and these resources are oversubscribed at least locally, if not globally. These applications include Air Force Satellite Control Network access scheduling (AFSCN) (Barbulescu *et al.* 2004b) and scheduling fleets of earth of observing satellites (Frank *et al.* 2001; Globus *et al.* 2003). Other applications, such as scheduling *individual* EOS satellites (Potter and Gasch 1998) are multi-capacity, oversubscribed problems when considering resources such as the solid state recorder (SSR).

The approach of Frank et al. to the EOS fleet scheduling problem is to apply a greedy constraint based scheduling algorithm, varied stochastically for improvement using Bresina's HBSS algorithm (Bresina 1996). This method employs a sophisticated contention heuristic, which encompasses both task priority and task contention, scheduling the observation with the highest contention in the time slot with the least contention. In an aggregate sense this approach should be successful in allocating a good number of high priority missions. No guarantee is made, though, for maintaining the state of scheduled missions as the HBSS algorithm schedules and reschedules from scratch in order to improve on the global objective.

(Khatib *et al.* 2003) explore a different facet of this same (EOS) domain: schedule revision and repair to contend with scheduling failures, changing viewing conditions, and targets of opportunity. The authors present an interleaved schedule execution and revision algorithm that dynamically decides which observations of lower utility to bump from the SSR in order to accommodate newer, high value observations. They consider lookahead techniques for bumping wisely and efficiently, but don't consider the possibility of rescheduling bumped observations.

A strong case is made by Barbulescu et. al. (Barbulescu *et al.* 2004a; 2004b) that a Genetic Algorithm solution performs better than either constructive or repair-based approaches for the Air Force Satellite Control Network (AFSCN) scheduling problem. The experiments they present focus on generating a maximally subscribed schedule from scratch, though, and do not address issues of task addition and schedule repair.

In (Kramer and Smith 2003; 2004) we specifically address the issues of task addition and schedule repair as they pertain to the USAF Air Mobility Command (AMC) scheduling problem (Becker and Smith 2000). We begin from the perspective that schedules in an oversubscribed domain can usually be generated efficiently to incorporate most of the high priority tasks, and it is not that difficult to decide which tasks to bump or which constraints to relax to incorporate more tasks. The harder problem is inserting additional tasks into an already oversubscribed schedule, *without bumping others* and while retaining the stability of the existing schedule.

The techniques applied to the AMC problem are transferable to problems in space mission scheduling with similar characteristics: oversubscription, multi-capacity resources, and some degree of flexibility in task placement (flexible time windows or resource pools). Before looking at several of these applications in more detail, we first summarize the AMC problem and the MissionSwap procedure as applied to it.

3 The AMC Scheduling Problem

Without loss of generality the AMC scheduling problem can be characterized abstractly as follows:

- A set T of tasks (or missions) are submitted for execution. Each task $i \in T$ has an earliest pickup time est_i, a latest delivery time lft_t, a pickup location $orig_i$, a dropoff location $dest_i$, a duration d_i (determined by $orig_i$ and $dest_i$) and a priority pr_i
- A set Res of resources (or air wings) are available for assignment to missions. Each resource $r \in Res$ has capacity $cap_r \geq 1$ (corresponding to the number of contracted aircraft for that wing).
- Each task i has an associated set Res_i of feasible resources (or air wings), any of which can be assigned to carry out i. Any given task i requires 1 unit of capacity (i.e., one aircraft) of the resource r that is assigned to perform it.
- Each resource r has a designated location $home_r$. For a given task i, each resource $r \in Res_i$ requires a positioning time $pos_{r,i}$ to travel from $home_r$ to $orig_i$, and a de-positioning time $depos_{r,i}$ to travel from $dest_i$ back to $home_r$.

A schedule is a *feasible* assignment of missions to wings. To be feasible, each task i must be scheduled to execute within its $[est_i, lft_i]$ interval, and for each resource r and time point t, ***assigned-cap***$_{r,t} \leq cap_r$. Typically, the problem is over-subscribed and only a subset of tasks in T can be feasibly accommodated. If all tasks cannot be scheduled, preference is given to higher priority tasks. Tasks that cannot be

placed in the schedule are designated as *unassignable*. For each unassignable task i, $pr_i \leq pr_j, \forall j \in Scheduled(T)$: $r_j \in Res_i \wedge [st_j, et_j] \cap [est_i, lft_i] \neq \emptyset$, where r_j is the assigned resource and $[st_j, et_j]$ is the scheduled interval.

Both the scale and continuous, dynamic nature of the AMC scheduling problem effectively preclude the use of systematic solution procedures that can guarantee any sort of maximal accommodation of the tasks in T. The approach adopted within the AMC Allocator application instead focuses on quickly obtaining a good baseline solution via a greedy priority-driven allocation procedure, and then providing a number of tools for selectively relaxing problem constraints and incorporating as many additional tasks as possible (Becker and Smith 2000). The task swapping procedure of (Kramer and Smith 2003) is one such schedule improvement tool.

4 The Basic Task Swapping Procedure

The task swapping procedure summarized below takes the solution improvement perspective of iterative repair methods (Minton *et al.* 1992; Zweben *et al.* 1994) as a starting point, but manages solution change in a more systematic, globally constrained manner. Starting with an initial baseline solution and a set U of unassignable tasks, the basic idea is to spend some amount of iterative repair search around the "footprint" of each unassignable task's feasible execution window in the schedule. Within the repair search for a given $u \in U$, criteria other than task priority are used to determine which task(s) to retract next, and higher priority tasks can be displaced by a lower priority task. If the repair search carried out for a given task u can find a feasible rearrangement of currently scheduled tasks that allows u to be incorporated, then this solution is accepted, and we move on to the next unconsidered task $u' \in U$. If, alternatively, the repair search for a given task u is not able to feasibly reassign all tasks displaced by the insertion of u into the schedule, then the state of the schedule prior to consideration of u is restored, and u remains unassignable. Conceptually, the approach can be seen as successively relaxing and reasserting the global constraint that higher priority missions must take precedence over lower priority missions, temporarily creating "infeasible" solutions in hopes of arriving at a better feasible solution.

In the subsections below, we describe this task swapping procedure, and the heuristics that drive it, in more detail.

4.1 Task Swapping

Figure 1 depicts a simple example of a task u that is unassignable due to prior scheduling commitments. In this case, u requires capacity on a particular resource r, and the time interval $ReqInt_{r,u} = [est_u - pos_{r,u}, lft_u + depos_{r,u}]$ defines the "footprint" of u's allocation requirement. Within $ReqInt_{r,u}$, an allocation duration *alloc-dur*$_{r,u} = pos_{r,u} + d_u + depos_{r,u}$ is required. Thus, to accommodate u, a subinterval of capacity within $ReqInt_{r,u}$ of at least *alloc-dur*$_{r,u}$ must be freed up.

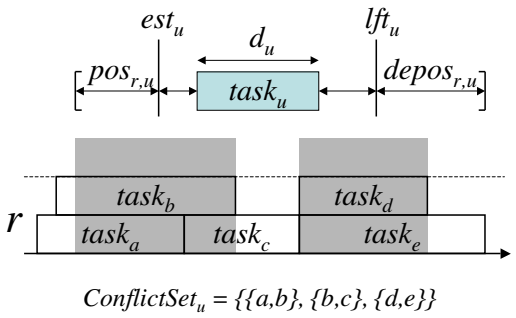

Figure 1: An unassignable task u and the conflict set generated for all intervals on resource, r.

To free up capacity for u, one or more currently scheduled tasks must be retracted. We define a conflict $Conflict_{r,int}$ on a resource r as a set of tasks of size Cap_r that simultaneously use capacity over interval int. Intuitively, this is an interval where resource r is currently booked to capacity. We define the conflict set $ConflictSet_u$ of an unassignable task u to be the set of all distinct conflicts over $ReqInt_{r,u}$ on all $r \in R_u$. In Figure 1, for example, $ConflictSet_u = \{\{a, b\}, \{b, c\}, \{d, e\}\}$.

Given these preliminaries, the basic repair search procedure for inserting an unassignable task, referred to as **MissionSwap**, is outlined in Figure 2. It proceeds by computing $ConflictSet_{task}$ (line 2), and then retracting one conflicting task for each $Conflict_{r,int} \in ConflictSet_{task}$ (line3). This frees up capacity for inserting $task$ (line 5), and once this is done, an attempt is made to feasibly reassign each retracted task (line 6). For those retracted tasks that remain unassignable, **MissionSwap** is recursively applied (lines 7-10). As a given task is inserted by **MissionSwap**, it is marked as protected, which prevents subsequent retraction by any later calls to **MissionSwap**.

In Figure 3, the top-level **InsertUnassignableTasks** procedure is shown. Once **MissionSwap** has been applied to all unassignable tasks, one last attempt is made to schedule any remaining tasks. This step attempts to capitalize on any opportunities that have emerged as a side-effect of **MissionSwap's** schedule re-arrangement.

4.2 Retraction Heuristics

The driver of the above repair process is the retraction heuristic **ChooseTaskToRetract**. In (Kramer and Smith 2003), three candidate retraction heuristics are defined and analyzed, each motivated by the goal of retracting the task assignment that possesses the greatest potential for reassignment:

MissionSwap($task, Protected$)
1. $Protected \leftarrow Protected \cup \{task\}$
2. $ConflictSet \leftarrow$ ComputeTaskConflicts (task)
3. $Retracted \leftarrow$ RetractTasks ($ConflictSet, Protected$)
4. **if** $Retracted = \emptyset$ **then** Return($\emptyset$) ; failure
5. ScheduleTask($task$)
6. ScheduleInPriorityOrder($Retracted$, least-flexible-first)
7. **loop for** $(i \in Retracted \wedge status_i = unassigned)$ **do**
8. $Protected \leftarrow$ MissionSwap($i, Protected$)
9. **if** $Protected = \emptyset$ **then** Return($\emptyset$) ; failure
10. **end-loop**
11. Return($Protected$) ; success
12. **end**

RetractTasks($Conflicts, Protected$)
1. $Retracted \leftarrow \emptyset$
2. **loop for** $(OpSet \in Conflicts$ **do**
3. **if** $(OpSet - Protected) = \emptyset$ **then** Return($\emptyset$)
4. $t \leftarrow$ ChooseTaskToRetract($OpSet - Protected$)
5. UnscheduleTask(t)
6. $Retracted \leftarrow Retracted \cup \{t\}$
7. **end-loop**
8. Return($Retracted$)
9. **end**

Figure 2: Basic MissionSwap Search Procedure

- **Max-Flexibility** - One simple estimate of this potential is the scheduling flexibility provided by a task's feasible execution interval, $ReqInt_{r,i}$, defined earlier. Let *feas-dur*$_{r,i}$ be the duration of $ReqInt_{r,i}$. Then an overall measure of task i's temporal flexibility is defined as[1]

$$Flex_i = \frac{\sum_{r \in Res_i} \textit{alloc-dur}_{r,i}}{\textit{feas-dur}_{r,i}}$$

leading to the following retraction heuristic:

$$MaxFlex = i \in C : Flex_i \leq Flex_j \forall j \neq i$$

where $C \in ConflictSet_u$ for some unassignable task u.

- **Min-Conflicts** - Another measure of rescheduling potential of a task i is the number of conflicts within its feasible execution interval, i.e. $|ConflictSet_i|$. This gives the following heuristic:

$$MinConf = i \in C : |ConflictSet_i| \leq |ConflictSet_j| \quad \forall j \neq i$$

where $C \in ConflictSet_u$ for some unassignable task u.

[1]This formulation varies from the one presented in our earlier work (Kramer and Smith 2003; 2004) where we compute the denominator of $Flex_i$ as $lft_i - est_i$, which did not take into account the duration of positioning and depositioning legs (setup duration). Given that the positioning durations vary by resource, but are constant per resource, a heuristic that takes them into account should be more informed. Our latest experiments have borne this out.

InsertUnassignableTasks($Unassignables$)
1. $Protected \leftarrow \emptyset$
2. **loop for** $(task \in Unassignables)$ **do**
3. SaveScheduleState
4. $Result \leftarrow$ MissionSwap($task, Protected$)
5. **if** $Result \neq \emptyset$
6. **then** $Protected \leftarrow Result$
7. **else** RestoreScheduleState
8. **end-loop**
9. **loop for** $(i \in Unassignables \wedge status_i = unassigned)$ **do**
10. ScheduleTask(i)
11. **end-loop**
12. **end**

Figure 3: InsertUnassignableTasks procedure

- **Min-Contention** - A more informed, contention based measure is one that consider's the portion of a task's execution interval that is in conflict. Assuming that dur_C designates the duration of conflict C, task i's overall contention level is defined as

$$Cont_i = \frac{\sum_{C \in ConflictSet_i} dur_C}{\sum_{r \in Res_i} \textit{feas-dur}_{r,i}}$$

leading to the following heuristic:

$$MinContention = i \in C : Cont_i \leq Cont_j, \forall j \neq i$$

4.3 Experimental results

The original experiments of (Kramer and Smith 2003) (carried out on a suite of 100 problems) demonstrated the efficacy of the **MissionSwap** procedure in the target domain. In this study, max-flexibility was shown to be the strongest performer; its application enabled **MissionSwap** to schedule, on average, 42% of the initial set of unassignable missions. Min-contention, scheduled 38%, but was almost three times slower. Min-conflicts proved less effective, scheduling only 30% on average.

These results were expanded upon in later work (Kramer and Smith 2004), where several search pruning techniques were tested on a similar suite of 100 problems. These techniques demonstrated that the **MissionSwap** algorithm could be sped up significantly without sacrificing solution quality. With these pruning techniques in place, the min-contention retraction heuristic outperforms max-flexibility in solution quality, however the latter retained a significant runtime advantage.

Several iterated stochastic techniques were applied to the best solutions found in the 100-problem set, and it was shown that further modest gains in solution quality were achievable given a reasonable amount of time.

5 How does MissionSwap work?

The **MissionSwap** task swapping procedure is clearly a general mechanism for schedule repair in the presence of multi-

capacity resources. While the AMC Allocator performs well employing a max-flexibility or min-contention heuristic, even random choice can be used although with higher cost and worse results. The max-flexibility heuristic, it should be noted, turns out to be well informed because of a reasonable variance in slack (more specifically the ratio of runtime to feasible window) across the set of input tasks. In domains where slack is not very variable, it is likely that a contention-based heuristic would be the better informed.

The important thing to note is that **MissionSwap** is *not* designed to swap a task into the schedule at the expense of other tasks. It is specifically designed to insert a task into the schedule by shifting a limited number of contending tasks, either to an alternative resource or to an alternate time in a manner that satisfies all task and resource constraints.

The power of **MissionSwap**, then, relies not so much on deciding where to schedule an unassignable task, but on deciding which tasks to unschedule, such that all will have opportunities to reschedule. If all do not reschedule, the algorithm searches recursively, but eventually all retracted tasks must schedule or the algorithm fails. Clearly, then, wise choice a retraction heuristic is important to the procedure, although construction of this heuristic is domain dependent and may require some tuning.

6 Applying MissionSwap to Space domains

We feel that **MissionSwap** is applicable generally to the problem of scheduling tasks in oversubscribed, multi-capacity and multi-resource space domains. Many of these domains experience high volatility as viewing conditions change and targets of opportunity make themselves known. At the same time that new tasks need to be inserted in a schedule, there is high value to maintaining as much stability as possible in the existing schedule. Of prime importance is not sacrificing high priority tasks at the expense of lower priority ones. Many of the existing approaches that we discussed earlier are not appropriate as "anytime" techniques, either rescheduling from scratch or making schedule repairs that sacrifice one task for another.

We now consider a few space scheduling problems and explore how the **MissionSwap** procedure might be applied in these contexts to handle oversubscription in a controlled way.

6.1 EOS Fleet Scheduling

Scheduling observations on an Earth Orbiting Satellite is a very challenging problem. There are often more observations to be allocated than can be accommodated by scientific instruments or data storage capacity. The domain is subject to many complex constraints and constant change due to local weather patterns and unpredictable events such as volcanic eruptions.

(Pemberton 2000) tackles the complexity of the EOS fleet domain by segmenting the scheduling problem into subsets of tasks ordered by priority, scheduling the subsets individually and then combining the schedules. This technique is not amenable to schedule repair, but Pemberton remarks that it could be combined with local repair to improve suboptimal schedules. Both (Frank *et al.* 2001) and (Globus *et al.* 2003) point out that scheduling the EOS fleet collectively can mitigate problems of complexity by tackling local resource oversubscription by shifting tasks from a resource on one satellite to a compatible resource on another. Frank et al. present a constraint-based approach to the problem, involving stochastic heuristic search.

Globus et al. compare the performance of a number of techniques as applied to both the single and two-satellite EOS scheduling problem including genetic algorithms, simulated annealing, squeaky wheel optimization, and stochastic hill climbing. Their preliminary experimental results point to simulated annealing as the most effective method, and that scheduling two satellites as a combined resource enabled scheduling of 28% more images.(Globus *et al.* 2003)

All of the above efforts aim at satisfying the objective of scheduling as many high priority observations as possible. They are not geared, however, to the task of inserting additional tasks in an oversubscribed EOS fleet schedule.

6.2 Comparison of the AMC and EOS Fleet Scheduling domains

The AMC and EOS Fleet Scheduling domains have both similarities and differences.

- *Task Priority.* In both problems there is a high premium placed on respecting task priority, however in the EOS domain individual task priority can vary as viewing conditions change.(Frank *et al.* 2001)
- *Task Flexibility.* In the AMC domain there is a great variability in the amount of slack in which to schedule a given task. Some very short duration missions can schedule feasibly in a very large duration window. Conversely, some long duration missions are highly constrained as to when they can schedule. In the EOS domain, observations are more discretely constrained as to when they can schedule. That is, due to the nature of the EOS orbit, an image can be acquired only during orbits when the satellite is positioned over the earth to capture the target or when the scientific instrument can be pointed to take the image (given that the instrument is pointable). This means that feasible time slots for a given observation may be found in orbit $n, n + k, n + 2k, \ldots$, and there may be some latitude for moving start times of observations within a given orbit. When we consider a fleet of satellites, this same observation may also be schedulable on other satellites in orbits $m, m+j, m+2j, \ldots$, which could describe very different intervals in time than the original satellite.
- *Task Linking.* One other aspect of the EOS problem that contrasts with AMC pertains to the dependencies between

observations. An AMC mission is an ordered sequence of individual legs, but the mission is scheduled as a whole. Only the positioning and depositioning legs and allocation durations change as alternate resource assignments (wings) are allocated.[2] In the EOS domain, individual images, or scenes, may be executed completely independently of one another, however there is a strong benefit to linking geographically contiguous scenes into a cluster. This not only saves on header and trailer data overhead, but may reduce the amount of warm-up time on the imaging instrument.(Potter and Gasch 1998) In the discussion that follows we ignore the benefit of clustering, and focus on scheduling a single unassignable observation. Our approach is extensible to preserving and extending clusters by allocating them as blocks of observations.

- *Resource Capacity.* Lastly, when a mission is assigned to a wing in the AMC problem, there are typically a good number of units of capacity allocated in the same time interval (on the order of 5-50), whereas in EOS scheduling, each instrument can execute one observation at a time, but multiple satellites may be able to accommodate that observation.

6.3 TaskSwap for schedule improvement in EOS Fleet Scheduling

We can now describe a solution to the problem of inserting additional observations in an existing EOS Fleet Schedule. We take as input a feasible schedule S of observations O, where each $o \in O$ is scheduled at time t on a unique satellite $\sigma \in F$, an EOS Fleet. (Without loss of generality we assume one scientific instrument per satellite.) Informally, we say that an observation o is *unassignable* if there exists no time t in its domain of feasible viewing windows such that o can be assigned at t and no domain constraints are violated.

For a **MissionSwap**-like procedure in the EOS domain to be effective, it is not sufficient to retract one task from a $Conflict$, since a $Conflict$ as defined earlier will trivially always be a set of exactly one task, thus leaving no choice for a retraction heuristic. In the AMC domain, there will in general be more than one unit of capacity assigned at a given time, whereas in the EOS domain this is not the case, so in order for a retraction heuristic to be effective, we redefine the notions of $Conflict$ and $ConflictSet$.

To define a larger set of possibilities for the retraction heuristic to consider, we focus on identifying alternative sets of tasks that must be retracted to place a new task in a given EOS observation interval. In Figure 4 one conflicted task o is depicted. In this illustration we consider one unassignable interval i on a satellite σ. In general there will be other conflicted intervals on the satellite's timeline. For $task_o$ to be able to schedule in the interval bounded by its earliest and latest start times, est_o and lft_o, either $task_a$ and $task_b$ or $task_b$ and $task_c$ must be retracted. This leads

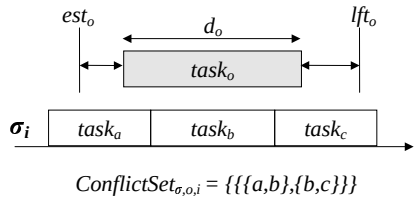

Figure 4: An unassignable observation o and the structured conflict set generated for one interval i on satellite σ.

us to a definition of a $RetractionSet$, the set of minimal critical sets (MCSs) (Laborie and Ghallab 1995) associated with the conflict. Let int be a conflicted interval for an unassignable observation o, if given an empty schedule, o is assignable in int, but given the current state of the schedule there is no subinterval in int of sufficient duration such that o can feasibly be assigned. Let $O^{'}$ be the set of observations that consume capacity over the interval int. Then the $RetractionSet$ for observation o over int is the set of all distinct subsets of $O^{'}$ such that unscheduling the observations in a subset would free up enough duration for o to schedule, and each subset is minimal. I.e., it is necessary to unschedule all observations in a subset to allow o to schedule.

We define a conflict, $Conflict_{\sigma,int}$, for a satellite σ in one interval int, as the $RetractionSet$ for int, and a $ConflictSet_{\sigma,o,int}$ as the set of distinct conflicts for o in int. Then $ConflictSet_o$ is the collection of $ConflictSet_{\sigma,o,int}$ over all conflicted intervals and satellites for o.

We can now define a procedure **TaskSwap** (Figure 5), which generalizes the **MissionSwap** procedure defined in Figure 2. The key difference between the **TaskSwap** and the **MissionSwap** procedures is that in the former the second argument to the procedure, $ProtectedSets$, tracks sets of observations (corresponding to MCSs) rather than individual observations. To protect a single observation rather than sets of observations would rule out that observation from being considered by **TaskSwap** as member of a different MCS. In addition, the call to **ComputeTaskConflicts** in line 2 now returns a set of $RetractionSets$.

To add an unassignable EOS observation $o^{'}$ to the schedule S, we call **TaskSwap**($o^{'}, \emptyset$) which will either fail to incorporate $o^{'}$ in S or generate a new feasible schedule $S^{'}$.

We must also redefine the function **RetractTasks** (See Figure 5). The main difference between **RetractTasks** and our earlier definition of **RetractTasks** in Figure 2 is line 4, where a call is made to Choose**Tasks**ToRetract (plural)

[2] Merging of missions – linking the end of one mission with the beginning of another – is allowed in the AMC domain, but this practice does not take place during routine scheduling, but can be initiated by the user as a post-processing phase.

TaskSwap($task, ProtectedSets$)
1. $ProtectedSets \leftarrow ProtectedSets \cup \{\{task\}\}$
2. $ConflictSet \leftarrow$ ComputeTaskConflicts (task)
3. $Retracted \leftarrow$ RetractTasks ($ConflictSet, ProtectedSets$)
4. **if** $Retracted = \emptyset$ **then** Return($\emptyset$) ; failure
5. ScheduleTask($task$)
6. ScheduleInPriorityOrder($Retracted$, least-flexible-first)
7. **loop for** $(i \in Retracted \wedge status_i = unassigned)$ **do**
8. $ProtectedSets \leftarrow$ TaskSwap($i, ProtectedSets$)
9. **if** $ProtectedSets = \emptyset$ **then** Return($\emptyset$) ; failure
10. **end-loop**
11. Return($ProtectedSets$) ; success
12. **end**

RetractTasks($ConflictSet, ProtectedSets$)
1. $Retracted \leftarrow \emptyset$
2. **loop for** $(Conflict \in ConflictSet$ **do**
3. **if** $(Conflict - ProtectedSets) = \emptyset$ **then** Return($\emptyset$)
4. $T \leftarrow$ ChooseTasksToRetract($Conflict - ProtectedSets$)
5. UnscheduleAll(T)
6. $Retracted \leftarrow Retracted \cup \{T\}$
7. **end-loop**
8. Return($Retracted$)
9. **end**

Figure 5: A Generalized Task Swapping Procedure

instead of Choose**Task**ToRetract (singular). The heuristic $ChooseTasksToRetract$ should select the subset of observations to unschedule that has the best chance of rescheduling. One such implementation is the $Contention_o$ heuristic – the contention for an observation o over all time slots – defined in (Frank *et al.* 2001). The computation would need to be aggregated over all observations in the conflict subset, and the subset of tasks with the minimum $Contention_o$ would be selected.

6.4 SSR Resource Allocation Problem

Data capacity on a satellite's Solid State Recorder (SSR) can often be the limiting factor in scheduling observations on an EOS. It is often possible to take more images than can be stored in the SSR before the next opportunity to downlink data occurs and the SSR capacity is replenished. (Potter and Gasch 1998) solve this with a multi-pass scheduling algorithm, which schedules first without regard to priority and on a second pass preempts lower priority tasks if resource capacities are overallocated. (Khatib *et al.* 2003) study the problem of incorporating new SSR tasks interleaved with execution in an onboard scheduler on an EOS satellite. Their method revises the schedule dynamically by sacrificing the task or tasks with lowest utility value to incorporate a new task of expected high utility value. Neither of these approaches considers the possibility of reallocating tasks on another SSR time block in order to incorporate new ones in an existing SSR schedule, however the **TaskSwap** Procedure we have just outlined for EOS Fleet Scheduling *is* applicable to SSR Resource Allocation.

6.5 Features of the SSR Capacity Model

The SSR is a multi-capacity resource, as its data volume is available for a number of observations to share at the same time. At certain intervals, the SSR capacity is freed up by downlinking observation data to ground stations. The SSR capacity model differs from the AMC model in that one conflicted observation may require that more than one observation be de-allocated from the SSR to free up necessary capacity.[3] In this sense, the SSR model is similar "vertically" in time as the EOS model is "horizontally." Figure 6 illustrates this.

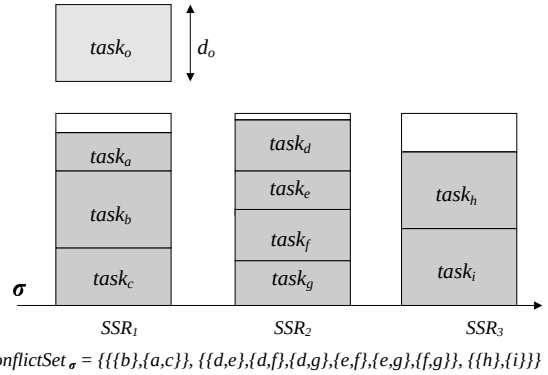

Figure 6: An unassignable observation o and the structured conflict set generated for several intervals of SSR capacity on satellite σ.

In this figure we have $task_o$ which requires d_o units of SSR data capacity. On satellite σ there are three periods of almost fully subscribed SSR usage, and we assume that the SSR data is downlinked in between these periods. (For now, we consider the one-satellite problem, but this discussion generalizes to multiple satellites.) For $task_o$ to schedule on σ, one of the SSR_i periods must free up at least enough capacity equal to $d_o - FreeSSR_i$, the free capacity on SSR_i. In this example, then, we define the $ConflictSet_\sigma$ to be the set of conflicts in Figure 6, the set of minimal subsets of tasks such that retracting one subset will allow $task_o$ to

[3] Technically speaking, the AMC model may require that more than one mission be freed up in any one time interval if, for instance, an unassignable mission requires three planes, while the scheduled missions only require one each. This differs from the SSR model in that chunks of SSR memory do not come in such discrete quanta, so that two observations using more capacity than minimally required may need to be re-allocated to schedule another.

schedule. I.e., the set of $RetractionSets$, similarly defined as in the EOS Fleet problem.

It should be noted that in the worst case the number of task combinations making up $ConflictSet_\sigma$ in this context can be exponential. In practice, this can be mitigated by using the pruning techniques that (Cesta *et al.* 2002) apply in sampling minimal critical sets.

6.6 TaskSwap for schedule improvement in SSR Scheduling

With this informal treatment of a $ConflictSet_\sigma$, and $ConflictSet_o$ applied for all $\sigma \in F$, it is not hard to see that the definition of **TaskSwap** for EOS scheduling applies as well to SSR scheduling. What is needed is an appropriate retraction heuristic for $ChooseTasksToRetract$, and again we find one in (Frank *et al.* 2001), $Contention_{r,o}$, which computes the contention for an observation o requiring an SSR resource r. This computation is aggregated over all observations in the conflict subset, and the subset of tasks with the minimum $Contention_{r,o}$ would be selected for retraction.

In actuality the SSR problem may be even more suited to the **TaskSwap** procedure than Figure 6 suggests. We have portrayed the SSR as only accommodating a few observations at a time, but on Landsat 7, for instance, typical usage is more like 90 observations (scenes).(Potter and Gasch 1998). This large number of observations to choose from could actually make the problem easier, particularly if there is a large variance in heuristic values. A small number of observations per SSR would mean a smaller search space, but it could be difficult to find large enough "holes" in the SSR schedule to fit one in, whereas a larger number of much smaller observations would generally provide candidates that could fit somewhere into SSR underutilization. Empirically this has been our experience in the AMC domain, where it is almost always possible to make space for a mission by rescheduling several smaller ones.

The SSR scheduling problem is, of course, intimately linked with the EOS observation scheduling problem described earlier, in that the observations are the same in both cases, only we consider their contention on different resources, the science instrument (or satellite, as we have couched it) and the SSR. De-conflicting an observation that is unassignable on one resource or the other we would attack the conflicted resource, while taking into account contention on other resources given the contention measure $Contention_{r,o}$ for that resource.

6.7 AFSCN scheduling

The AFSCN scheduling problem involves assigning user requests for communication from one of over 100 satellites to 16 antennas located on 9 ground stations around the Earth.(Barbulescu *et al.* 2004b) This problem has many of the same attributes as the AMC scheduling problem: multiple resources, multi-capacity resources, and oversubscribed resources. The approach that Barbulescu et al. take to managing oversubscription, though is very different. They have found that for a problem suite using actual AFSCN data, a genetic algorithm outperforms both constructive, repair-based, and local search in generating a schedule with the fewest unassigned tasks.(Barbulescu *et al.* 2004a) The strength of this approach is that there are some very striking characteristics of the problem domain, all of which would be difficult to encode in a heuristic function, but which the genetic algorithm is able to learn over many iterations. For instance, it is the case that simple one-for-one task swaps are usually not sufficient to resolve a conflict, whereas multiple moves must be made.(Barbulescu *et al.* 2004a)

This approach to managing oversubscription in a large problem in the presence of multiple, multi-capacity resources is clearly very exciting, as it is able to generate a highly allocated schedule in a short period of time – on the order of a few seconds for approximately 500 requests.[4] Furthermore, it holds out the promise of transferring to some of the other domains we have discussed, without the need for specially crafted domain heuristics.

On the other hand, it is not clear whether this approach is applicable to the problem of incorporating new requests into an existing schedule without disrupting good portions of that schedule. While this may not be a requirement of the AFSCN domain, it *is* a requirement of some of the other domains that we have studied, so it remains to be seen whether the genetic algorithm technique of Barbulescu et al. transfers to the problem of schedule improvement and schedule repair in those domains. It seems clear, though, that the **TaskSwap** procedure that we have explored for the EOS Fleet scheduling and SSR allocation domains *is* transferrable to the AFSCN problem.

6.8 Features of the AFSCN scheduling Capacity Model

The AFSCN capacity model is interesting in that it can be considered a hybrid between the AMC, EOS, and SSR models that we have studied. Typically an AFSCN request to schedule involves a particular antenna at a ground station during a visibility interval of the satellite to that ground station. However, alternate intervals at different groundstations, may also be requested.(Barbulescu *et al.* 2004b)

Given the math of 9 ground stations and 16 antennas, it is clear that some groundstations have more than one antenna, but others have only one. Those ground stations with more than one antenna can be considered as a multi-capacity resource somewhat analogous to an AMC wing (with much less capacity). A request on one antenna should be schedulable on another antenna at the same station during the same time interval. Ground stations with one antenna behave like

[4] Personal communication from L. Barbulescu.

a unit-capacity EOS satellite, as there are multiple discrete time windows per antenna in which to schedule. Finally, capacity usage on a given antenna is comparable to that on a single EOS solid state recorder in that the amount of capacity requested varies in proportion to the quantity of data downlinked.

6.9 TaskSwap for schedule improvement in AFSCN scheduling

Without getting into the details, then, it should be clear that a $ConflictSet$ can be defined utilizing the concept of a $RetractionSet$ in terms of requests, ground stations, antennas, and intervals, and that the **TaskSwap** procedure defined in Figure 5 is applicable to adding an unassignable communications request to an oversubscribed AFSCN schedule. An appropriate flexibility heuristic for requests suitable as a retraction heuristic for **RetractTasks** is cited in(Barbulescu *et al.* 2004a) based on the work of Gooley.(Gooley 1993) This heuristic encompasses some of the same features as the max-flexibility and min-contention heuristics employed by **MissionSwap** in the AMC domain.

7 Conclusions

There has been increasing interest in addressing multi-capacity and multi-resource problem domains in space. Existing techniques are often capable of doing a good job at generating a schedule in the face of oversubscription, however most techniques are not suited to the problem of task addition and schedule repair. Those techniques that do address schedule repair generally rely on bumping of lower priority tasks. We have presented a general purpose TaskSwap procedure and shown its applicability to adding tasks in an anytime fashion in three multi-capacity and multi-resource space scheduling domains.

The merit (or inapplicability) of **TaskSwap** to domains other than AMC will only be demonstrated with experiments in those domains. However, testing out new techniques in a domain as complex as EOS fleet scheduling, for instance, is difficult when not involved in that domain on a day-to-day basis. It is possible to abstract out the important characteristics of the domain to create a suite of problems for experimental purposes, but these abstract models often miss an important domain feature that could make the results of those experiments unreliable. Nevertheless, we are currently working on abstracting out the essential features of the AMC domain with a test data set to make it available to other researchers. We encourage others in the field to make problem instances and data available, and encourage them to experiment with applying **TaskSwap** to their domains. One such problem set that has recently come to our attention is the 2003 challenge problem of the French OR Society (ROADEF) on the Management of Missions of Earth Observing Satellites.[5]

[5]http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2003/challenge2003_en.html#sujet

There is certainly a good deal left to explore in this area. In the AMC domain itself, there are a number of variants to **TaskSwap** that we have tried, but which are outside the scope of this paper. Other techniques remain to be explored. In (Kramer and Smith 2004) we tested a number of pruning techniques to improve the performance of the algorithm, but one pruning method we have not yet experimented with is "resource pruning." Looking closely at the description of **MissionSwap** (Figure 2), **RetractTasks** is called on *all* alternate resources. The reason for this is that **MissionSwap** is agnostic as to which resource the unassignable task should be allocated to, so all resources are de-conflicted even though only one will actually be used. This has the synergistic effect of creating more flexibility for the retracted tasks to schedule. It is possible, though, that by selecting only "the best" resource to de-conflict (the one whose tasks are most likely to be able to reschedule), better quality solutions will be returned more quickly.

Acknowledgements

The authors would like to thank Robert Morris of NASA Ames Research Center and Laura Barbulescu of Colorado State University for discussions which helped us better understand the EOS and AFSCN scheduling domains and scheduling techniques. The work reported in this paper was sponsored in part by the Department of Defense Advanced Research Projects Agency (DARPA) and the US Air Force Research Laboratory under contracts F30602-00-2-0503 and F30602-02-2-0149, by the USAF Air Mobility Command under subcontract 10382000 to Northrop-Grumman Corporation, and by the CMU Robotics Institute.

References

L. Barbulescu, A. E. Howe, L.D. Whitley, and M. Roberts. Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS-04), to appear*, Whistler BC, June 2004.

L. Barbulescu, J. P. Watson, L. D. Whitley, and A. E. Howe. Scheduling space-ground communications for the airforce satellite control network. *Journal of Scheduling, to appear*, 2004.

M. A. Becker and S. F. Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proc. 5th Int. Conf. on AI Planning and Scheduling*, pages 32–41, Breckenridge CO, April 2000.

J. Bresina. Heuristic-baised stochastic sampling. In *Proceedings 13th national Conference on AI*, pages 271–278, Portland OR, 1996. AAAI Press.

A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8:109–136, 2002.

J. Frank, A. Jónsson, R. Morris, and D. E. Smith. Planning and scheduling for fleets of earth observing satellites. In *Proc. 6th Int. Symposium on AI, Robotics and Automation for Space*, 2001.

A. Globus, J. Crawford, J. Lohn, and A. Pryor. Scheduling earth observing satellites with evolutionary algorithms. In *International Conference on Space Mission Challenges for Information Technology*, 2003.

T. Gooley. Automating the satellite range scheduling process. Master's thesis, Air Force Institute of Technology, 1993.

M. D. Johnston and G. Miller. Spike: Intelligent scheduling of hubble space telescope observations. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.

L. Khatib, J. Frank, D.E. Smith, R. Morris, and J. Dungan. Interleaved observation execution and rescheduling on earth observing systems. In *ICAPS-03 Workshop on Plan Execution)*, Trento Italy, June 2003.

L. A. Kramer and S. F. Smith. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proceedings 18th International Joint Conference on Artificial Intelligence*, Acapulco Mexico, August 2003.

L. A. Kramer and S. F. Smith. Task swapping for schedule improvement, a broader analysis. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS-04), to appear*, Whistler BC, June 2004.

P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings 14th International Joint Conference on Artificial Intelligence*, Montreal, August 1995.

S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161–205, 1992.

J. C. Pemberton. Toward scheduling over-constrained remote-sensing satellites. In *Proceedings 2nd Int. NASA Workshop on Planning and Scheduling for Space*, San Francisco, CA, March 2000.

W. Potter and J. Gasch. A photo album of earth: Scheduling landsat 7 mission daily activities. In *Proc. of the International Symposium Space Mission Operations and Ground Data Systems*, 1998.

G. Rabideau, R. Knight, S. Chien, A. Fukanaga, and A. Govindjee. Iterative planning for spacecraft operations using the aspen system. In *Procedings 5th International Symposium on AI, Robotics and Automation for Space*, 1999.

G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. 12th National Conf. on AI*, Seattle WA, Aug 1994.

M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.