Using Adaptive Priority Weighting to Direct Search in Probabilistic Scheduling

Andrew M. Sutton and **Adele E. Howe** and **L. Darrell Whitley**
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
{sutton,howe,whitley}@cs.colostate.edu

Abstract

Many scheduling problems reside in uncertain and dynamic environments – tasks have a nonzero probability of failure and may need to be rescheduled. In these cases, an optimized solution for a short-term time horizon may have a detrimental impact over a broader time scale. We examine a scheduling domain in which time and energy on a phased array radar system is allocated to track objects in orbit around the earth. This domain requires probabilistic modeling to optimize the expected number of successful tasks on a particular day. Failed tasks must be attempted again on subsequent days. Given a set of task requests, we study two long-term objectives: percentage of requests initially successful, and the average time between successful request updates. We investigate adaptive priority weighting strategies that directly influence the short-term objective function and thus indirectly influence the long-term goals. We find that adapting priority weights based on when individual tasks succeed or fail allows a catalog of requests to be filled more quickly. Furthermore, with adaptive priorities, we observe a Pareto-front effect between the two long-term objectives as we modify how priorities are weighted, but an inverse effect of weighting when the priorities are not adapted.

Introduction

Real world scheduling problems often contain a large degree of uncertainty. These problems usually require some form of probabilistic modeling (Berry 1993) to find solutions that optimize some predictive measure. In the scheduling literature, uncertainty typically manifests as unknown task durations, uncertain resource availability over time, or uncertain release/due times.

In this paper we examine an orbit track scheduling domain in which the likelihood of a task *succeeding* changes over time. In this domain, the objective is to allocate time and energy on a phased array radar in order to fill and update a catalog of tracking requests for objects in orbit. For each day, a given tracking request specifies a set of tasks that must execute on the radar to collect information about a particular object. An object in the catalog is "updated" if the majority of its tracking tasks are successful. A request is "filled" if its object updated at least once.

In the short-term, we can use a probabilistic modeling technique to generate schedules that maximize the expected number of successful tasks for a particular day. If a request is not filled at all, it should be attempted on a later day. Furthermore, requests that are updated *more frequently* will produce higher quality data since dynamic orbital information is ephemeral. The two long term goals can be defined as 1) the percentage of catalog that is successfully filled after a number of days, and 2) the frequency of updates on requests that have been filled.

Finding an optimal solution to the short-term goal is NP-hard (Sutton, Howe, & Whitley 2007). Therefore, in order to tractably maximize the expected number of successful tasks in a day, we must apply heuristic methods. However, these solutions will have an impact on the long-term goals. To maximize the percentage filled, requests that are not yet filled should have priority over those that have already been successful. We develop a strategy in which priorities are adapted to reflect the current state and contribute in turn to the short-term objective optimized by metaheuristic search.

We hypothesize that adaptive priority weighting can overcome the myopia of the short term objectives directing search. We find that this is indeed the case for percentage of catalog filled; adaptive priority weighting can obtain a 5% increase in the number of filled requests by the long-term horizon. We also discover an interesting trade-off between the two long-term goals: an increase in filled requests causes the mean frequency of updates to decrease.

The remainder of this paper is organized as follows. In the next section we discuss the motivation for this work and give a formal problem description. We then present our empirical analysis of the adaptive priority weighting. Finally we discuss our findings.

Motivation: scheduling orbit tracks

The amount of earth-orbiting material is quickly approaching a critical level (Stansbery 2004; Broad 2007). It has therefore become increasingly important to meticulously detect and track objects in space such as defunct satellites, debris, and operational orbital devices. The Space Surveillance Network (SSN) is a heterogeneous array of sensing devices developed for this purpose. The SSN is responsible for maintaining a catalog of objects in orbit to prevent potential collisions with space missions, predict orbital decay, and facilitate space/ground communication.

Each constituent of the SSN is responsible for performing observations on objects that move through its field of view. The sensing site receives a set of tracking requests from North American Aerospace Defense Command (NORAD) and must allocate its resources over the course of the day in order to fill these requests and perform updates on the space catalog data.

In this research we consider a particular component of the SSN: the AN-FPS/85 phased array radar at Eglin Air Force Base in Florida, USA[1]. A phased array radar is capable of steering a beam of radio energy almost instantaneously using phase cancellation among a pattern of antennas on its face plane. By interleaving pulses, the phased array system can track several objects at once subject to energy constraints. The probability that an object is successfully illuminated is a function of where the object lies in space and how it is moving with respect to the radar system.

We are given a catalog of tracking requests that specify tracking tasks for given objects. Each tracking request r corresponds to a particular orbiting object and is formally defined as

$W(r)$	a set of visibility windows (or *passes*)
$d(r)$	duration of requested track
$h(r)$	a flag that specifies how often to track

The duration specifies how long the tracking task(s) must be to collect the needed observations. The h flag is set to $true$ when the requested object must only be tracked *once* a day. Otherwise the object must be tracked in every pass. Each request also has a corresponding *priority* that expresses its relative importance. In this paper we will study how these priorities can be manipulated to serve long-term goals.

Each pass in the set $W(r)$ can be defined by a tuple of four maps (est, lft, e, p). In particular, the k^{th} pass $(1 \leq k \leq |W(r)|)$ for a tracking request r consists of

$est(r,k)$	earliest start time of track
$lft(r,k)$	latest finish time of track
$e(r,k)[t]$	energy requirement at time t
$p(r,k)[t]$	probability of success at time t

where $est(r,k)$ is the time when the object for r penetrates the array's field of view to begin its k^{th} pass and tracking can begin. Similarly, $lft(r,k)$ is when the object completes its k^{th} pass and exits the field of view. At this point tracking can no longer occur.

The energy requirement $e(r,k)$ is a function of the object's *range* with respect to the radar array. Objects which are further away require more energy to illuminate than objects that are nearby. Since the range is changing dynamically, the energy requirement may vary in different passes, or even at different times during the same pass. The energy requirement is calculated using a *waveform catalog*: a mapping from object range intervals to energy needed to illuminate when the radar is pulsing.

The probability of success $p(r,k)$ is also a time-dependent quantity. The SNR profile is a function of both the object's range and angle off the *boresight* direction: the normal vector to the array's face plane. In particular, if θ is the boresight angle at time t and $range$ denotes the object's range at time t, the SNR profile is computed as follows.

$$snr(r)[t] = \frac{\sigma \cdot \cos^2 \theta}{range^4}$$

Where σ is the *radar cross section*: the nominal reflective surface area of the object. This SNR profile is related to the likelihood of successful illumination. We define the probability of success as

$$\begin{aligned} p(r,k)[t] &= \frac{1}{1+\exp\left(\frac{-snr(r)[t']}{3}+8\right)} \\ \text{where } t' &= t + \frac{d(r)}{2} \end{aligned}$$

This corresponds to the object's probability of success during the *middle* of a tracking task if it were to begin at time t.

We characterize a *schedule* for a particular day as a family X of decision variable sets $X(r)$ for each request r. Each decision variable $x(r,k) \in X(r)$ corresponds to the tracking time assigned to request r in its k^{th} pass. The decision variables are programmed subject to constraints. For each request r, we must find a set:

$$X(r) = \begin{cases} x(r,k) \neq \emptyset & \text{for } \textit{some } k \in \{1, \ldots, |W(r)|\} \\ & \text{if } h(r) = true \\ x(r,k) \neq \emptyset & \text{for } \textit{all } k \in \{1, \ldots, |W(r)|\} \\ & \text{if } h(r) = false \end{cases}$$

subject to

$$est(r,k) \leq x(r,k) < x(r,k) + d(r) \leq lft(r,k) \qquad (1)$$

and

$$\sum_{r \in M_t} e(r,k)[t] \leq E[t] \quad \text{for all times } t \qquad (2)$$

where $M_t = \{r | x(r,k) \leq t \text{ and } x(r,k) + d(r) \geq t\}$, that is, the set of all tasks scheduled to be tracked during time t and $E[t]$ is the system energy constraint at time t. These constraints ensure that tracking times are scheduled to occur only when the corresponding objects are visible, and that the total energy required for tracking at any given time does not exceed available system energy.

The quantity $p(r,k)$ allows us to use a probabilistic modeling technique to construct schedules that maximize the *expected* number of successful tracks. We denote the *expected sum successful* as

$$ESS(X) = \sum_{x(r,k) \in X} p(r,k)[x(r,k)] \qquad (3)$$

i.e., the expected value of the number of successful tasks at the end of the day. A scheduling algorithm must find a decision variable programming X to maximize this quantity subject to the constraint equations (1) and (2).

Scheduling framework

One scheduling approach might be to assign all tracking tasks to occur at the time that gives the highest SNR profile during a pass. If all tasks can be feasibly inserted, this

[1]We thank the Eglin SLEP program at ITT industries for providing us with information.

gives the optimal expected sum successful. However, since the system is significantly oversubscribed, we are forced to choose a feasible subset of these tasks. Furthermore, we have found that in a highly constrained environment, allowing tasks to be scheduled in suboptimal positions can create schedule packings with a *higher* overall expected sum of successful tasks (Sutton, Howe, & Whitley 2007).

This allows us to define two scheduling paradigms for this domain. *On-peak* scheduling, in which tasks are only placed on their "peak" probability of success in a pass, and *relaxed* scheduling in which tasks are allowed to be placed anywhere within the pass. Under sufficient constraints, maximizing ESS using either approach is NP-hard (Sutton, Howe, & Whitley 2007). This motivates the use of metaheuristic search algorithms to find schedules that maximize ESS.

Schedule builders

We follow a common practice by encoding solutions as permutations of tasks and employing a deterministic schedule builder that considers tasks in the order they appear in the permutation (Whitley 1989; Syswerda & Palmucci 1991; Bierwirth 1995; Barbulescu *et al.* 2004b). We define two schedule builders that set decision variables differently: *on-peak* and *relaxed*.

Let ω be a permutation of tasks. Initially we start with an empty schedule. The on-peak schedule builder attempts to place each task ω_i at the point in the pass that gives the highest probability of success. In particular, suppose ω_1 corresponds to the task requested by tracking request r in pass k. The *peak* time m for a pass is the time that gives the highest probability of success within that pass. Specifically, $m = \max_t p(r,k)[t]$ for $t \in [est(r,k), lft(r,k)]$. Then if $e(r,k)[m] < E[m]$, the schedule builder sets $x(r,k) \leftarrow m$ and $E[m] \leftarrow E[m] - e(r,k)[m]$. Otherwise, the task is discarded. Then the next element ω_2 is considered, and so on. The post condition imposed by the on-peak schedule builder is that all tasks in the created schedule are both time and resource feasible, and every included task lies on its peak probability of success.

The relaxed schedule builder allows tasks to be placed away from their peak probability of success. For each permutation element, first an on-peak insertion is attempted as above. If this fails, the first feasible time closest to the peak (if any) is located. If no feasible times exist for a task, it is discarded, and the next element of the permutation is considered.

Simulator

We decompose the day into disjoint one-hour short-term horizons we call "scheduling periods" and generate a solution to each period separately. The permutation elements are simply ordinal numbers of the tasks that are visible to the radar during the corresponding period. We refer to this set of tasks as *active* tasks.

After each scheduling period, the solution found by search is executed by a simulator. This simulator considers the schedule and assigns "success" or "failure" to each executed task based on the probability of success at its assigned tracking time.

The simulator marks a request as *updated* if over half of its requested passes are successfully tracked during the course of the day. We define a request as *filled* if it has been updated at least once. Since requests with $h(r) = true$ require only *one* successful tracking task, each such request that has at least one successful pass during the scheduling period is removed from consideration in later scheduling periods for the remainder of the day. Note that if multiple passes occur during the short-term horizon it is possible that more than one pass will be successful for a request with $h(r) = true$.

Consider a single day in which no requests have yet been filled. Because of time and energy constraints and task failures, not all requests in the space catalog will be filled by end of the day. This leaves the catalog incomplete. Furthermore, orbit decay due to atmospheric drag, collisions, and secular gravitational effects from the sun and moon render older observations obsolete. If updates are made on a particular object several times over the course of many days, it will generally have higher quality data in the catalog.

Our two long term goals are the *percentage of the catalog filled*, and the *mean number of days an object goes without being updated*. A schedule that optimizes the expected sum successful may actually be detrimental to either or both of these goals. Indeed, by definition of ESS, requests with passes that have, on average, higher probabilities of success will tend to starve out those with lower probabilities.

Approaches to scheduling

We apply metaheuristic search to generate solutions that maximize the expected number of successful tasks for the short-term horizon. Since such solutions tend to be myopic, we use adaptive priority weighting to respond to the success and failures of requests.

In this way, our strategy is a hybrid of the *proactive* and *reactive* scheduling paradigms defined by Davenport and Beck (2000). Cesta, Policella, and Rasconi (2006) have pointed out that scheduling under uncertainty can often be solved by an integration of both proactive and reactive approaches. In proactive scheduling, uncertainty is addressed at schedule creation time. On the other hand, in reactive scheduling, decisions are made on-line in response to the current system state. Our search algorithms thus belong to the proactive classification as they search the space of short-term schedules for more robust potential solutions prior to schedule execution. Between schedule days, we adapt priorities in response to the long-term state of the system: a strategy that is more congruous with the reactive archetype.

Search algorithms

The search algorithms explore the space of *active* task permutations and generate solutions using one of the schedule builders. Note that the mapping between permutations and schedules is not bijective. Consider two permutations ω and ω' that are identical apart from positions i and j being swapped, i.e. $\omega_i = \omega'_j$ and $\omega_j = \omega'_i$. If tasks i and j are not in contention, then, for either schedule builder, both ω and ω' may result in identical schedules. This suggests a

name	description	builder
OP-SLS	On-Peak Stochastic Local Search	on-peak
RX-SLS	RelaXed Stochastic Local Search	relaxed
RX-GEN	RelaXed GENITOR	relaxed

Table 1: Metaheuristic search algorithms and their descriptions.

search space with a large number of "plateaus": connected components on which the schedule evaluation is identical. Stochastic local search has been effectively applied as a way of moving quickly across plateaus (Mitchell & Forrest 1997; Barbulescu *et al.* 2004a; Hoos & Stützle 2004).

We list the search algorithms in Table 1. The stochastic local search algorithms, *on-peak stochastic local search* (OP-SLS) and *relaxed stochastic local search* (RX-SLS), start with an initial random permutation and iteratively apply the *swap* move operator. Let a and b be two distinct, randomly generated integers such that, without loss of generality, $1 \leq a < b \leq n$. The swap move operator maps a permutation ω to a permutation ω' by transposing the order ω_a and ω_b. A *neighbor* permutation, ω' is obtained from ω by setting all $\omega'_i = \omega_i, i = 1, \ldots, a-1, a+1, \ldots b-1, b+1, \ldots n$ and $\omega'_a = \omega_b, \omega'_b = \omega_a$. The *relaxed GENITOR* (RX-GEN) is an implementation of the GENITOR genetic algorithm (Whitley 1989) and evolves a population of permutations, using the relaxed schedule builder to evaluate their fitness.

Adaptive priority weighting

On a particular day, if a request has not yet been filled, its corresponding tasks become more important than those that belong to filled requests. To that end, we develop an adaptive priority strategy. To each request r we assign a priority $\pi(r)$. Each time a request is updated, its priority is decreased. If a request is not updated during the day, its priority is increased.

To weight the impact of priorities in the objective function we define a weight function α that maps priorities to scalar weights. Higher priority tasks receive a higher α-weighting than lower priority tasks. We define the *priority-weighted expected sum successful* as a modified variant of the original objective function defined in Equation (3).

$$PWESS(X) = \sum_{x(r,k) \in X} p(r,k)[x(r,k)] \cdot \alpha(\pi(r)) \qquad (4)$$

This objective incorporates information about whether or not a task belongs to a request that has been filled recently. Consider a simplified case in which all tasks have an equal probability of success. If these two tasks are in contention, the one that has not been updated for longer is selected for inclusion since it will have a higher priority.

Suppose we have priorities $\pi(r) \in \{1, 2, \ldots, m\}$ where 1 is the highest priority and m is the lowest. Given a bias function f, we define α as

$$\alpha(\pi(r)) = w \frac{f(m - (\pi(r) - 1))}{\sum_{i=1}^{m} f(i)}$$

We normalize by the sum so that, no matter what choice of bias function, the sum of the priorities is equal.

The priority assignments partition the set of tracking requests into m disjoint equivalence classes. A particular task's contribution to the $PWESS$ is the product of its probability of success and its α weight. The w term sets the maximum value for the α weight. The bias function adjusts the weight differential between priority classes.

We experiment with five different bias functions.

$$\begin{array}{rcll}
f_{con}(x) & = & 1/m & \text{constant} \\
f_{log}(x) & = & \ln x & \text{logarithmic} \\
f_{linear}(x) & = & x & \text{linear} \\
f_{quad}(x) & = & x^2 & \text{quadratic} \\
f_{exp}(x) & = & \exp(x) & \text{exponential}
\end{array}$$

Note that the f_{con} bias is essentially equivalent to no priority weighting at all.

To assess the contribution of priority alone, we introduce two new scheduling algorithms that operate on priority alone. *On-peak greedy* (OP-GREEDY) simply sorts the active tasks by priority order and uses the on-peak schedule builder to insert them into the schedule. *Relaxed greedy* (RX-GREEDY) uses the same sorting, but employs the relaxed schedule builder.

Experiments

Using real data from NORAD[2], we generated a tracking request catalog consisting of 1000 actual objects that passed over the Eglin site over seven days in March 2007. We used an SGP orbit propagator (Hoots & Roehrich 1980) to compute passes, SNR data, and ranges for each object.

Each schedule day is decomposed into 24 disjoint one-hour scheduling periods (short-term horizons). Tasks that are visible to the radar during the scheduling period are marked as active and considered for inclusion into the schedule. We ran each algorithm during each scheduling period to generate a set of feasible decision variables for that period. These solutions are then executed on the simulator.

We added a noise element to the simulator in order to emulate real fluctuations in actual SNR profile. The *noisy* probability of success $p'(r, k)$ is used by the simulator to determine whether or not a task is successful is defined as follows.

$$p'(r,k) = p(r,k) + \epsilon$$

where ϵ is a zero-mean Gaussian random variable with a variance proportional to the radar cross section of the object for r. This conforms to the fact that available radar cross section data are aggregate estimates, and often objects have undulating cross sections as they spin or tumble through space.

We set the long-term horizon at seven days. Correspondingly, we used seven priority classes (to give any request the chance to rise to a priority 1 by the end of the long-term horizon). On the first day, each request is assigned an initial uniform random priority. This simulates "starting the process" on an arbitrary day. At the end of each schedule day, the successfully tracked passes are tallied.

[2]A comprehensive on-line catalog of SSN orbital data is maintained at `http://www.space-track.org`

We designed a factorial experiment as follows.

Independent variables

1. *Non-adaptive vs. adaptive.* Does adapting the priorities have an effect on long-term goals?
2. *Bias function.* Does the choice of bias function have an effect? We used the five different bias functions for α-weighting: constant, logarithmic, linear, quadratic, and exponential. For the w term we used 10.0.
3. *Priorities.* Do the priorities contribute to the effect somehow? To observe the impact of the priorities alone, we also used a neutral priority strategy in which no priorities exist.

Dependent variables

1. *Percent of catalog filled.* At the end of the long-term horizon, we assess what percentage of the 1000 objects have had at least one update.
2. *Mean days without update.* We count how many days each request goes without an update, and take the average over the entire catalog at the end of the long-term horizon. We assume that no updates have occurred on any object prior to the designated start day. When we calculate this mean, we exclude the requests that are never filled.

To collect statistics, for each combination of independent variables we ran ten trials out to the long-term horizon. We used the following search algorithm settings. Our stopping criterion for both stochastic local search and the genetic algorithm was 10000 evaluations of the schedule builder in each one-hour scheduling period. The genetic algorithm used a population size of 100 and a selective pressure of 2.0.

Results

Non-adaptive vs. adaptive. We record the two response variables at the end of each trial and report each one as a point on a two dimensional plot in Figure 1. Due to space constraints, we only report the results using the linear bias function (see next section for the effect of bias). Note that there is a clear separation between the two strategies, allowing us to draw a partition around each. When using the adaptive strategy, each algorithm clearly gains an advantage in percentage of catalog filled while suffering an increase in mean days without updates.

Across all trials, no algorithm was able to fill the catalog completely. We performed a post-hoc analysis on the data set and found that 42 objects have a maximum probability of success of less than 0.2. It is therefore not unreasonable to assume that at least 4.2% of the requests will be statistically unlikely to be filled using any of our scheduling algorithms in our trials. This creates a ceiling effect which we represent on the plots as a broken horizontal line at 95.8%. In practice, these objects would be treated specially, a point we shall revisit in the discussion.

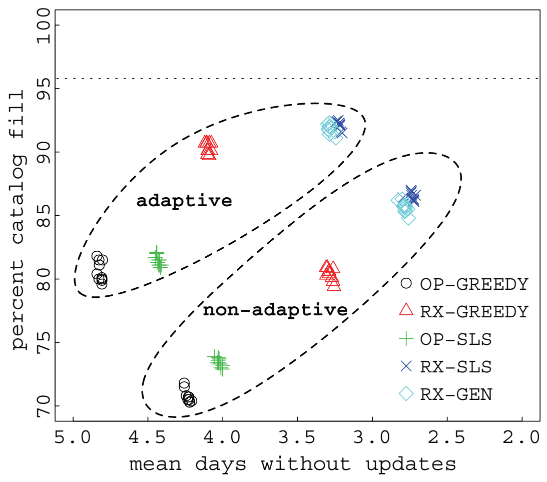

Figure 1: Results from 10 trials on each algorithm for adaptive and non-adaptive strategies. Horizontal line drawn at 95.8 (percentage of requests with $p(r,*) < 0.2$). Note that dominating solutions appear toward the top right.

Effect of bias function. We report the trials across all bias functions for all algorithms in Figure 2. The greedy algorithms appear as data clouds (with RX-GREEDY clearly dominating in both metrics). This is not surprising since the greedy algorithms do not consider the priority-weighted objective function, and simply schedule each task in priority order in a weight invariant manner.

We observe an interesting development in the search algorithms. The effect of the different bias functions appear to be distributed across a Pareto-front between the two objectives. We see an asymptotic increase in percentage of catalog filled with each bias function and a marked decrease in mean days without update. This suggests an interesting trade-off which will be discussed below.

Effect of priorities Some of the observed differences may be due to the priorities themselves. To control for this, we consider the performance of search without any priorities. The results are similar for all search algorithms. Due to space constraints, we report the results from just RX-SLS in Figure 3. Unsurprisingly, the constant bias function has the same effect as no priorities at all. The composite plot shows the movement away from this point by the adaptive and non-adaptive strategy. Clearly, the non-adaptive strategy for bias functions other than constant is dominated in both objectives by the solutions generated without priorities.

To examine the statistical significance of this relationship, we performed two sample t-tests between each strategy in each dimension. The results from these tests (again for RX-SLS) are reported in Tables 2 and 3. In Table 2 we find that the *no-priority* strategy performs best in terms of mean days without update. On the other hand, Table 3 shows that the adaptive strategy significantly obtains a higher catalog fill by the long-term horizon.

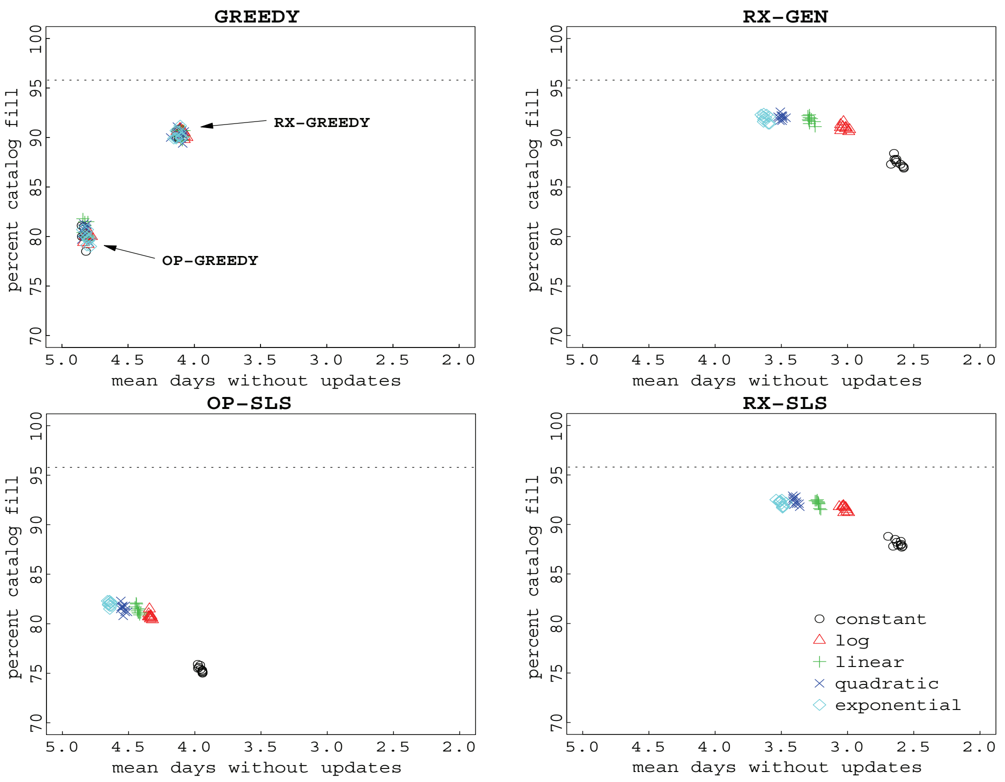

Figure 2: Effect of bias function in adaptive priority strategy across all algorithms.

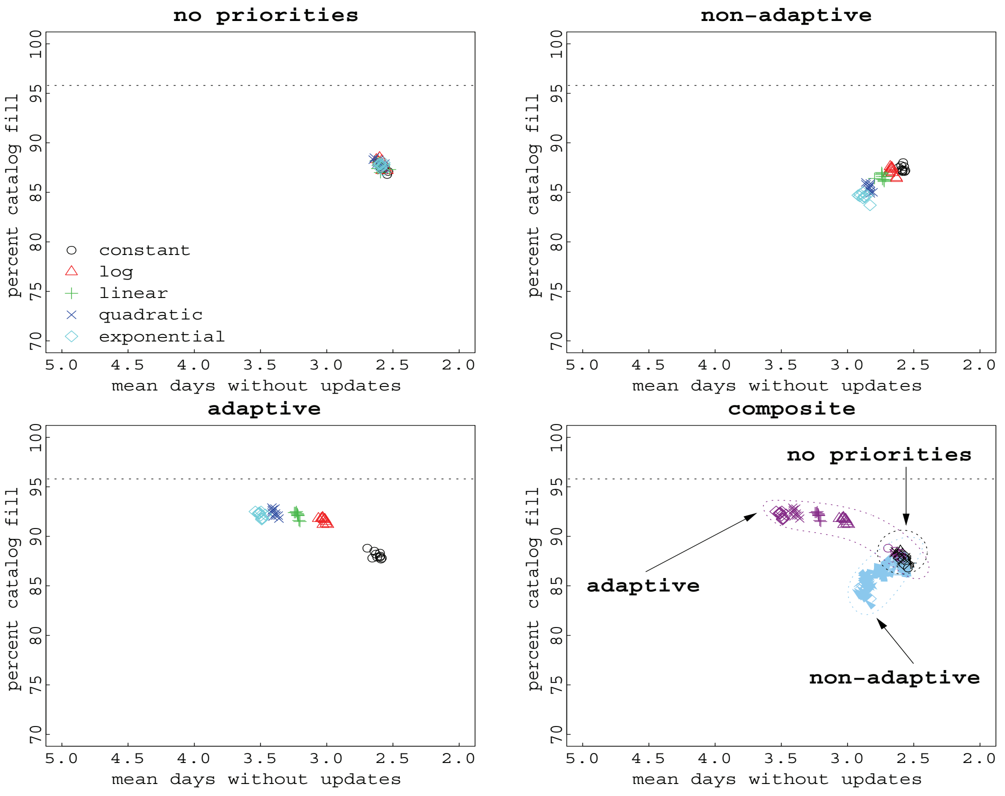

Figure 3: Effect of priority on a representative algorithm (RX-SLS). Fourth plot is a superposition of the former three (labeled partitions are drawn around each group corresponding to the plot from which it came).

Mean days without update		
test	t	p
no-priority < adaptive	-12.64	< 0.00001
non-adaptive < adaptive	-8.70	< 0.00001
no-priority < non-adaptive	-9.52	< 0.00001

Table 2: Left-tailed two-sample Welch t-tests for priority experiments: mean days without update at long-term horizon.

Percentage of catalog filled		
test	t	p
adaptive > no-priority	14.80	< 0.00001
adaptive > non-adaptive	17.89	< 0.00001
no-priority > non-adaptive	8.93	< 0.00001

Table 3: Right-tailed two-sample Welch t-tests for priority experiments: percent catalog filled by long-term horizon.

Post-hoc analysis: Requests unfilled. As we previously mentioned, no strategy or algorithm completely filled the catalog by the long-term horizon. A small percentage (4.2%) of these requests had maximum probability of success of 0.2 or less. However, we find that this did not account for all the remaining requests at the long-term horizon.

We performed an analysis on the requests that each algorithm left unfilled and found that some indeed had quite high probability of success. Table 4 reports statistics for each algorithm and each bias function using the adaptive strategy. In particular we observe the mean number of requests left unfilled by day 7 over all 10 trials. We also report the number of these requests with a *minimum* success probability of greater than or equal to 0.8 over all seven days and the number with *maximum* success probability of less than or equal to 0.2.

The algorithms that are better performers in terms of percentage of catalog filled seem to be leaving fewer high probability requests behind. However, the reduction in low probability requests is not as dramatic. This is evidence that algorithms that produce more efficient schedule packings will tend to reduce the number of *high-probability* unfilled requests at the long-term horizon.

There is a fairly large collection of requests left unfilled that lie on the spectrum between our "low" and "high" probability classifications. We conjecture that a combination of both low success probability and high schedule contention contributes to these failures.

Discussion

Our experiments show that an adaptive priority strategy can be used to influence a short-term objective function to drive solutions that affect long-term goals. We find that the percentage of catalog filled by the long-term horizon can be increased by adapting priority weights for the objective functions optimized by search. However, we find a trade-off in the mean number of days without update. Indeed, when priorities are adapted, the solutions lie along a Pareto-front. As we increase the weight differential between priority classes, we find an asymptotic increase in the percentage of catalog filled with a significant increase in mean number of days without update. This effect can be explained fairly intuitively. In order to diversify the set of objects updated, *individual* updates must become less frequent. On the other hand, to decrease the mean days without update, a strategy can focus on a small, easy to detect subset of the request catalog to drive the mean down.

alg	bias	UF	HP	LP
OP-GREEDY	const	198.8	26.5	40.8
	log	200.2	26.9	40.3
	linear	194.1	25.9	40.2
	quad	194.3	24.1	41.1
	exp	201	26	41.2
RX-GREEDY	const	96.6	4.9	38.4
	log	95.6	5.1	39.2
	linear	96.7	4.9	38.7
	quad	97.3	5.2	38.4
	exp	97	4.8	38.9
OP-SLS	const	246.1	21.1	41
	log	192.4	12.6	41.1
	linear	185.7	9.9	41.3
	quad	184.6	9.3	41.2
	exp	179.8	9.4	41.1
RX-SLS	const	119.1	5	38
	log	84.1	1.9	38.1
	linear	78.8	1.4	38
	quad	76.3	1.8	38.1
	exp	78.4	1.3	38.1
RX-GEN	const	125.2	5.7	38.9
	log	90	3	38.1
	linear	82	2.4	38
	quad	79.5	2.1	38
	exp	80.6	1.8	38.3

Table 4: Statistics of unfilled requests at long-term horizon. UF is number of unfilled requests. HP denotes the number *high probability* requests: i.e., those with minimum probability greater than or equal to 0.8. LP denotes the number of *low probability* requests: those with maximum probability less than 0.2 over the entire data set.

There appears to be an inverse effect of weighting when priorities are *not* adapted (c.f. Figure 3). We conjecture this is evidence of a *starvation* effect. Tasks that belong to high priority requests will contribute more to the short-term objective function and therefore tend to be scheduled more often and receive more updates. As the relative weights increase, the influence that priority exerts on the objective increases and the starvation effect becomes more expressed.

One interesting effect we observe is the dramatic separation between algorithms (c.f. Figure 1). To explore this effect, we look at the mean packing of the schedules: the number of individual tracking tasks that were feasibly inserted into the schedule over the course of the seven day period. Furthermore, we examine what percentage of these individual tracking tasks were "hit" (successfully collected the needed observations) and what percentage were "missed" by the simulation. We collect the means of these data over all trials, bias functions, and priority strategies and report the results in Table 5.

Two striking features are evident in this table. First, the

algorithm	packing	% hit	% missed
OP-GREEDY	4659.59	87.4922	12.5078
RX-GREEDY	7351.5	82.7308	17.2692
OP-SLS	5005.07	92.9927	7.00729
RX-SLS	8421.88	90.9114	9.08863
RX-GEN	8309.21	90.8033	9.1967

Table 5: Mean schedule packing, mean percentage of tasks hit (successful) and mean percentage of tasks missed (unsuccessful) for each algorithm. The means are taken over all trials, all bias functions, and all priority strategies.

relaxed algorithms tend to obtain a higher packing on average than the on-peak algorithms. This is an effect of the relaxed schedule builder which allows more options for task insertion. Second, the search algorithms tend to have more successful tasks and fewer missed tasks on average. The search algorithms are clearly leveraging the expected sum successful objective in the short-term horizons to produce schedules with higher success likelihoods. Unsurprisingly, the *on-peak* local search has the highest mean percentage of successful tasks and lowest mean percentage of unsuccessful tasks: a consequence of the on-peak schedule builder which places each task at the time that gives it the highest probability of success. This comes at an expense to schedule packing.

We found a small percentage of requests that were never successful due to aberrantly low SNR profiles. These profiles typically correspond to very small and/or distant objects. In practice, these objects are tracked by allocating a large percentage of system energy into dedicated radio pulses. In our experiments, we wanted to control the resource usage by decoupling it from probability of success. In the future, we will experiment with changing energy allocation to affect success probability.

Conclusions

In this paper we examined a strategy that employs adaptive priority weighting to drive search to produce short-term schedules that directly influence long-term goals. We found that this priority adaptation can increase the percentage of a request catalog filled out at an expense to the mean number of days a request goes without being updated. We see a 5% increase in the cardinality of the filled request set by the end of the long-term horizon. For our small data set, this amounts to approximately 50 requests in 7 days. However, if we assume that the effect scales with catalog size, we can extrapolate for larger request sets. With a catalog of 10,000 objects, for example, this translates to a gain of 500 extra objects a week. Such a catalog size is roughly consistent with recent numbers of tracked objects: typically around 13,000 (Stansbery 2004).

We also showed that the expansion in percentage of requests filled tends to promote a decrease in mean number of days without update. Since this metric corresponds to data currency, an increase in successful requests could potentially cause a corresponding increase in stale data. In future work we will investigate this relationship more thoroughly.

Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-07-1-0403. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

Barbulescu, L.; Howe, A. E.; Whitley, L. D.; and Roberts, M. 2004a. Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *Proceedings of the International Conference on Planning and Scheduling*.

Barbulescu, L.; Watson, J.-P.; Whitley, L. D.; and Howe, A. E. 2004b. Scheduling space-ground communications for the air force satellite control network. *J. Scheduling* 7(1):7–34.

Berry, P. 1993. Uncertainty in scheduling: Probability, problem reduction, abstractions and the user. In *Proceedings of the IEE Colloquium on Advanced Software Technologies for Scheduling*.

Bierwirth, C. 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum* 17(2):97–92.

Broad, W. J. 2007. Orbiting junk, once a nuisance, is now a threat. *The New York Times* CLVI(53847):F1.

Cesta, A.; Policella, N.; and Rasconi, R. 2006. Coping with Change in Scheduling: Toward Proactive and Reactive Integration. *Intelligenza Artificiale*. (Accepted).

Davenport, A. J., and Beck, J. C. 2000. A survey of techniques for scheduling with uncertainty. Unpublished.

Hoos, H. H., and Stützle, T. 2004. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufman Publishers.

Hoots, F. R., and Roehrich, R. L. 1980. Models for propagation of NORAD element sets. Spacetrack Report No. 3.

Mitchell, M., and Forrest, S. 1997. Fitness landscapes: Royal road functions. In Bäck; Fogel; and Michalewicx., eds., *Handbook of Evolutionary Computation*, volume B2.7. Institute of Physics Publishing. 1–25.

Stansbery, E. G. 2004. Growth in the number of SSN tracked orbital objects. Technical Report IAC-04-IAA.5.12.1.03, NASA Johnson Space Center.

Sutton, A. M.; Howe, A. E.; and Whitley, L. D. 2007. Exploiting expectation trade-off in probabilistic modeling with stochastic local search. Journal of Scheduling (in submission).

Syswerda, G., and Palmucci, J. 1991. The application of genetic algorithms to resource scheduling. In Belew, R. K., and Booker, L. B., eds., *Proceedings of Int'l Conf. on Genetic Algorithms (ICGA)*, 502–508. Morgan Kaufmann.

Whitley, L. D. 1989. The GENITOR algorithm and selection pressure. In *Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufman.