Advising busy users on how to cut corners

Pradeep Varakantham, Stephen F. Smith
TR #CMU-RI-08-17
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA, 15213
{pradeepv,sfs}@cs.cmu.edu

May 12, 2008

Abstract

In this paper, we consider the problem of assisting a busy user in managing her workload of pending tasks. We assume that our user is typically oversubscribed, and is invariably juggling multiple concurrent streams of tasks (or work flows) of varying importance and urgency. There is uncertainty with respect to the duration of a pending task as well as the amount of follow-on work that may be generated as a result of executing the task. The user's goal is to be as productive as possible; i.e., to execute tasks that realize the maximum cumulative payoff. This is achieved by enabling the assistant to provide advice about where and how to shed load when all tasks cannot be done.

A simple temporal problem with uncertainty and preferences (called an STPPU) provides a natural framework for representing the user's current set of tasks. However, current STPPU solution techniques are inadequate as a basis for generating advice in this context, since they are applicable only in the restrictive case where all pending tasks can be accomplished within time constraints and our principal concern is support in oversubscribed circumstances. We present two techniques for solving the oversubscribed STPPU problem. Given an ordering of tasks, these algorithms identify which tasks to ignore, which to compress and by how much, to maximize quality. We show experimentally that our approaches perform significantly better than techniques adapted from prior research in oversubscribed scheduling.

1 Introduction

Researchers have been developing automated personal assistants in many walks of life, ranging from office environments [5] to therapy planning [14] to disaster rescue scenarios [17]. A key problem in most of these domains is task management: users typically need to shuffle between multiple concurrent streams of tasks of varying importance and urgency to meet deadlines, expedite task outcomes, and maximize productivity. Task management is challenging for several reasons: (a) there are numerous un-controllable factors that influence the task duration (variable complexity, varying user capability etc.), and hence uncertainty is inherent to task duration; (b) tasks are frequently not independent but instead situated within larger workflows (e.g., accomplishing a project, scheduling a meeting) that introduce various temporal dependencies between tasks, (c) there is often uncertainty with respect to the set of follow-on tasks that might be generated as a result of executing a given task and also with respect to task durations; and (d) quality varying based on how much time is spent on it. Because of all of these factors, it can be quite difficult to accurately project future workload over time, and even more difficult to decide which subset of tasks to execute and how much time to devote to each in circumstances where projected workload is too great to accomplish within time constraints.

Task Management has been studied rather extensively in the literature [19, 3, 5, 6]. Some of this work has focused on understanding the potential value of automated support to human users, while other work concentrated on tracking task progress, recognizing task boundaries and managing task context. The majority of work aimed at providing task management guidance has assumed deterministic models and focused on task ordering, task allocation and task delegation decisions. In contrast, our research attempts to operate from a more realistic underlying model that incorporates knowledge of both relevant temporal constraints on task execution (dependencies, deadlines) and task uncertainty. Starting from this model, we focus specifically on the problem of dealing with oversubscribed situations, where the user has more to do than available time will allow.

We model the task management problem as a scheduling problem, i.e. as a problem of computing if and when each task in the user's list of pending tasks should be performed, given known constraints. Hence, the quality of the advice that is extracted (e.g., a reduced to do list) is directly proportional to the quality of the schedule itself. We use a variation of the Simple Temporal Problem with Preferences and Uncertainty (STPPU) model [16] as a basic representation of the problem. STPPU is a good fit for our problem because it simultaneously captures temporal dependencies, durational uncertainty and preferences (i.e., quality profiles) over tasks.

However, existing approaches for solving STPPUs [16] do not provide leverage in an oversubscribed setting, since they provide no information unless a complete solution is found. The objective in an oversubscribed setting is to determine the best partial solution - a schedule incorporating the subset of tasks that maximizes overall solution quality. We provide two linear (quadratic) programming based techniques that compute this schedule given a task ordering.

Given the NP-hard complexity of this problem, our techniques are primarily heuristic. However, with our first technique, we can establish provable guarantees on the solution quality in restricted settings. To provide a complete framework for providing task management advice and to exploit opportunities arising at run-time, we couple the offline techniques with an online re-scheduler, and empirically evaluate their performance in a sample task management domain. We compare performance to two baseline procedures representative of prior deterministic and reactive approaches to solving oversubscribed scheduling problems. Our techniques are shown to consistently outperform these techniques and illustrate the potential power of an STPPU based approach.

2 Task Management Problem

We formalize the task management problem as follows: $\langle \Gamma, \{\Lambda_\tau\}, \{\Upsilon_{\tau,\tau'}\}, \{g_{\kappa,\kappa'}\}\rangle$. Γ represents the set of pending tasks and τ represents a task in Γ. Λ_τ is the task model for a task τ and is denoted by the tuple $\langle \kappa, [lb_\tau, ub_\tau], z_\tau, f_\tau\rangle$:
(a) κ denotes task type.
(b) Task Duration is selected from the interval $[lb_\tau, ub_\tau]$.
(c) Duration uncertainty is specified by the random variable z_τ, with mean, μ_τ and deviation, σ_τ. Its value is selected by the environment from $[\mu_\tau - \sigma_\tau, \mu_\tau + \sigma_\tau]$ at run-time.
(d) Quality profile for a task is $f_\tau(X_\tau)$. It represents the quality obtained as a function of the effort (time spent on the task by the user). Depending on the domain, the actual quality profiles can be arbitrarily complex. However, it has been shown extensively in machine learning [20] that generalized linear models[1] represent complex settings very well (Occam's Razor principle offers a good intuition). Hence, we assume linear profiles: $f_\tau(X_\tau) = aX_\tau + b$, where X_τ is task duration. Note that our techniques are also applicable in settings with (a) piecewise linear and concave profiles (due to quality maximization) : $f_\tau(X_\tau) = \min\{a_1X_\tau + b_1, a_2X_\tau + b_2, \cdots\}$; and (b) quadratic concave profiles: $f_\tau(X_\tau) = aX_\tau^2 + bX_\tau + c$.

$\Upsilon\tau, \tau'$ denotes a temporal dependency between tasks τ and τ': $\langle \gamma, [lb_{\tau,\tau'}, ub_{\tau,\tau'}]\rangle$. γ represents the dependency type, i.e. whether it is between the start of τ and start of τ' or start of τ and end of τ' etc. These dependencies could either be decision variables (controllable) or selected by the environment (uncontrollable) and can be used to capture deadline constraints. $g_{\kappa,\kappa'}$ denotes the generate dependency between task types, κ and κ'. It represents the probabilistic number of tasks of type κ' generated from executing a task of type κ (e.g. "process an email" task generates a "schedule a group meeting" task) .

The objective is to compute a schedule, which incorporates a subset of all tasks in Γ and does not violate any constraints, while maximizing quality.

3 STPPU and Modeling task management

The Simple Temporal Problem with Preferences and Uncertainties (STPPU) model of [16] is represented using the tuple (E_c, E_u, C_c, C_u), where E_c represents the set of controllable events, E_u represents the set of un-controllable events (values set by the environment), C_c is the set of controllable constraints, and C_u is the set of uncertain or contingent constraints. Constraints in C_c are characterized by the tuple: $\langle lb_{ij}, ub_{ij}, f_{ij}\rangle$, where f_{ij} represents the quality profile over the duration interval $[lb_{ij}, ub_{ij}]$, for the edge between controllable events ev_i and ev_j. Alternatively, constraints in C_u

[1](a) Learning these linear models has been studied quite extensively [20] and we have a learning framework that extends the above work to learn quality profiles in our domain. In fact, we are able to learn all the attributes in the task model.
(b) Head/tail regions of quality profiles where quality decreases over time can be ignored, because scheduling the task for a higher duration provides lower quality, e.g. Gaussian profiles.

are denoted by $\langle lb_{ij}, ub_{ij}, p_{ij} \rangle$, where $ev_i \in E_c \cup E_u$, $ev_j \in E_u$ and p_{ij} is the probability density function over the interval $[lb_{ij}, ub_{ij}]$.

The task management problem of the previous section maps directly into this model. Each task τ is represented by three events: two certain events, τ_s and τ_d, that represent the task's start and desired end time; and one uncertain event, τ_e that represents the actual end of τ. A certain constraint, $\langle lb_\tau, ub_\tau, f_\tau \rangle$ is introduced to capture the task duration (τ_s to τ_d link) and an uncertain constraint, $\langle \mu_{z_\tau} - \sigma_{z_\tau}, \mu_{z_\tau} + \sigma_{z_\tau}, P(z_\tau) \rangle$ (where $P(z_\tau)$ represents the probability density function for z_τ) is added to represent durational uncertainty (τ_d to τ_e link).Generate dependencies are not encoded directly in the STPPU model, but are instead applied upfront to expand Γ to include expected tasks.

To enhance the scalability of our techniques, we augment our basic representation of tasks in the extended task set Γ with a higher-level representation of Γ where tasks of the same type (e.g., email processing tasks associated with specific flagged messages) are aggregated into a single super task (e.g., process flagged email), and we assume further that all temporal dependencies between tasks of different types can be expressed at the super task level. A given super task is represented in the same manner as a base level task, with corresponding composite duration constraints (both certain and uncertain). [2]

4 Computing how to cut corners

As mentioned earlier, our goal is to compute a schedule that incorporates a subset of tasks such that the overall quality is maximized. Given a task ordering procedure(e.g. [18, 1, 10]), the core problem is to find a subset of tasks to be eliminated. Since the number of tasks is discrete and quality profiles of tasks are independent, solving this problem requires solving an integer optimization problem (0/1 knapsack) and hence is NP hard.

In this section we present two techniques for solving this oversubscribed STPPU problem. Both assume an overall solution structure where the aggregate (super task) level problem is first solved by relaxing the integer variables and producing a continuous solution. If the problem is indeed over-subscribed, this continuous solution will dictate some amount of compression of one or more super tasks, and in each case this compression may actually entail shedding of some fractional number of tasks (e.g., it may specify that 2.5 tasks of a process-email super task originally composed of 5 base tasks should be ignored). To compute the actual tasks that need to be shed, the technique is recursively applied to the individual tasks within each super-task, employing the super task level solution as an overall deadline. Since there are no dependencies, this is a simpler problem. In essence, we are finding a solution to the relaxed problem and then finding the nearest integer solution.

Both of the techniques we present solve the STPPU offline. They output "policies"

[2]Note that this abstract level of representation is not as restrictive as one might initially think, since multiple super tasks can be introduced for different subsets of tasks of the same type for those subsets that might more naturally be partitioned (e.g., sequences of email tasks corresponding to different subject threads).

(along with the desired shedding) for setting the values of executable time points (the start and end time of tasks), which are contingent on occurrence of earlier uncertain events (duration uncertainty on earlier tasks). We refer to the set of uncertain events or contingent edges occurring before an executable event as its dependency set. As the STPPUs are constructed based on expected set of tasks, the schedules computed at run-time need to be modified to account for deviations from the expected behavior.

4.1 Linear Policy Advice, LPA

In this technique, we impose a linear structure on policies employed for setting values to the executable time points in the STPPU. This idea was first used by [13] to solve STNUs (Simple Temporal Network with Uncertainty). In our case, this assumption is used to solve oversubscribed STPPUs, a more challenging problem owing to two factors: optimizing with respect to quality and oversubscription. We compute a (offline optimized) dynamically controllable schedule (that incorporates a sub-set of tasks), because this schedule will be used for advising a user and hence must be robust against uncertainty.

A linear policy for an event (say starting event of task τ) is characterized by the following equation:

$$t_s^{\tau} = c_0^{\tau,s} + \sum_{\tau' \in D_\tau} c_{\tau'}^{\tau,s} z_{\tau'} \tag{1}$$

where t_s^{τ} is the value assigned to the starting event of τ, D_τ is the dependency set of τ (same for starting and ending events of a task), $c_{\tau'}^{\tau;s}$ are coefficients in the linear policy, $z_{\tau'}$ are random variables associated with contingent edges in D_τ.

We need to determine coefficients, $c_{\tau'}^{\tau;s}$, which will maximize the quality while not violating the temporal constraints. Algorithm 1 provides the optimization problem for LPA. The first key aspect of solving the oversubscribed STPPU is being able to optimize with respect to quality. Since this is offline optimization, we maximize the overall expected quality (expectation over the durational uncertainty), i.e. the sum of expected quality of all tasks. Expression for the expected quality of a task, τ for a linear quality profile is derived below:

$$E[f_\tau(X_\tau)] = \int P(D_\tau) f_\tau(X_\tau) dD_\tau,$$

where the integration is over the distribution of random variables in D_τ. Furthermore, since durational uncertainties of tasks are independent:

$$= \int_{z_a} P(z_a)... \int_{z_k} P(z_k) \int_{z_l} P(z_l) f_\tau(X_\tau) dz_l dz_k ... dz_a$$

If the duration of a task is X_τ, then for linear quality profiles (a and b are constants):

$$= \int_{z_a} P(z_a)... \int_{z_k} P(z_k) \int_{z_l} P(z_l)(aX_\tau + b) dz_l dz_k ... dz_a$$

Due to the linear policy assumption of eqn (1), $X_\tau = t^\tau_d - t^\tau_s = c^{\tau,d}_0 - c^{\tau,s}_0 + \sum_{\tau' \in D_\tau} (c^{\tau,d}_{\tau'} - c^{\tau,s}_{\tau'}) z_{\tau'}$. Thus, $aX_\tau + b$ will be of the form $\hat{c}^\tau_0 + \sum_{\tau' \in D_\tau} \hat{c}^\tau_{\tau'} z_{\tau'}$.

$$= \int_{z_a} P(z_a) ... \int_{z_k} P(z_k) \int_{z_l} P(z_l)(\hat{c}^\tau_0 + \sum_{\tau' \in D_\tau} \hat{c}^\tau_{\tau'} z_{\tau'}) dz_l dz_k ... dz_a$$

Since $\int_{z_l} P(z_l) dz_l = 1$ and $\int_{z_l} P(z_l) z_l dz_l = \mu_{z_l}$

$$= \int_{z_a} P(z_a) ... \int_{z_k} P(z_k)(\hat{c}^\tau_0 + \hat{c}^\tau_l \mu_{z_l} + \sum_{\tau' \in \{D_\tau - \{l\}\}} \hat{c}^\tau_{\tau'} z_{\tau'}) dz_k ... dz_a$$

$$= \int_{z_a} P(z_a) ... (\hat{c}^\tau_0 + \hat{c}^\tau_l \mu_{z_l} + \hat{c}^\tau_k \mu_{z_k} + \sum_{\tau' \in \{D_\tau - \{l,k\}\}} \hat{c}^\tau_{\tau'} z_{\tau'}) ... dz_a$$

Proceeding this way, the end result is

$$E[f_\tau(X_\tau)] = \hat{c}^\tau_0 + \sum_{\tau' \in D_\tau} \hat{c}^\tau_{\tau'} \mu_{z_{\tau'}} \tag{2}$$

Similarly, for quadratic profiles, the overall result is of the form

$$E[f_\tau(X_\tau)] = \hat{c}^\tau_0 + \sum_{\tau' \in D_\tau} (q^\tau_{\tau'})^2 \sigma^2_{z_\tau} + \sum_{\langle \tau', \tilde{\tau} \rangle \in D^2_\tau, \tau' \neq \tilde{tau}} r_{\tau'} r_{\tilde{\tau}} \mu_{z_{\tau'}} \mu_{z_{\tilde{\tau}}} + \sum_{\tau' \in D_\tau} \hat{c}^\tau_{\tau'} \mu_{z_{\tau'}} \tag{3}$$

Equation 2 or Equation 3 provide the objective function in the optimization problem used for solving an oversubscribed STPPU (Line 2 in Algorithm 1), depending on the type of quality profiles. In Equation 3, $\sigma^2_{z_\tau}$ denotes the variance of the durational uncertainty variable for task τ. However, for piecewise linear and concave functions, obtaining the expression for (expected) optimal quality is difficult. Hence, we can employ upper/lower bounds on the expected quality as the objective function and these can be computed by using gradient and conjugate function information corresponding to the quality profile and the probability density function [4].

Algorithm 1 LPASOLVER($\{\mu_{z_\tau}\}, \{\sigma_{z_\tau}\}$)

```
1: variables: $c^{\tau,s}_{\tau'}, c^{\tau,d}_{\tau'}, \delta_\tau; \forall \tau \in \Gamma, \tau' \in \{0\} \cup D_\tau$
2: max $\sum_{\tau \in \Gamma} f_\tau(X_\tau) + \mathcal{P}(\delta)$
3: **s.t.**
4: $\forall \tau \in \Gamma$
5: $X_\tau = c^{\tau,d}_0 - c^{\tau,s}_0 + \sum_{\tau' \in D_\tau} (c^{\tau,d}_{\tau'} - c^{\tau,s}_{\tau'}) \mu_{z_{\tau'}}$
6: $lb_\tau \leq c^{\tau,d}_0 - c^{\tau,s}_0 + \sum_{\tau' \in D_\tau} (c^{\tau,d}_{\tau'} - c^{\tau,s}_{\tau'})(\mu_{z_{\tau'}} \pm \sigma_{z_{\tau'}}) + \tilde{\delta}_\tau \leq ub_\tau$
7: $lb_{\tau,\tilde{\tau}} \leq c^{\tilde{\tau},s}_0 - c^{\tau,e}_0 + \sum_{\tau' \in D_\tau \bigcap D_{\tilde{\tau}}} (c^{\tilde{\tau},s}_{\tau'} - c^{\tau,e}_{\tau'})(\mu_{z_{\tau'}} \pm \sigma_{z_{\tau'}}) \leq ub_{\tau,\tilde{\tau}}$
8: $\tilde{\delta}_\tau = \delta_\tau + \sum_{\tau' \in \Gamma} g_{\tau',\tau} \delta_{\tau'}$
```

The second challenge involved in solving an oversubscribed STPPU is handling the oversubscription. We address this by providing a simple yet effective technique: introducing slack variables, δ which ensure that the duration constraints (reason for oversubscription) are never violated (line 6). These slack variables are assigned the required compression or shedding so that a solution exists. A key design parameter is the penalty term, $\mathcal{P}(\delta)$ in line 2 of Algorithm 1. The role of this term is to ensure that shedding happens (i.e. δ variables assigned a value greater than 0) only when no solution exists for the optimization problem of Algorithm 1. Furthermore, based on its design, this term can be used to compute a high quality solution with minimum shedding. Due to the generate dependencies, shedding a task results in duration compression of another task and this is captured through the constraint on line 8.

Addressing the first two challenges yields a STPPU solver, albeit one that is not entirely optimized. This is due to presence of $z_{\tau'}$ terms in line 5 and 6 of Algorithm 1. Constraints in lines 5 and 6 are to be satisfied only at the extremes of the random variable, i.e. $\mu_{z_\tau} \pm \sigma_{z_\tau}$. In a "non-oversubscribed" STPPU, this is fine. However, in an over-subscribed STPPU where shedding of tasks is performed, the uncertainty interval $[\mu_{z_\tau} - \sigma_{z_\tau}, \mu_{z_\tau} + \sigma_{z_\tau}]$ should be appropriately shortened to account for the task compression, i.e. $[\mu_{z_\tau} - \sigma_{z_\tau} - q1_\tau\delta_\tau + q2_\tau\delta_\tau, \mu_{z_\tau} + \sigma_{z_\tau} - q1_\tau\delta_\tau - q2_\tau\delta_\tau]$, where $q1_\tau$ and $q2_\tau$ are constants that capture the mapping between the mean and deviation of the current durational uncertainty (after shedding) and the original duration uncertainty, i.e. corresponding to the entire set of expected tasks. However, such a modification would introduce non-linearity in Algorithm 1, because there will be terms that are a product of variables ($c_{\tau'}$ and δ_τ) on lines 5 and 6. To address this, we introduce an iterative technique (Algorithm 2) that calls Algorithm 1 repeatedly until convergence.

Algorithm 2 ITERATIVESOLVER()

```
1: $\delta \leftarrow$ INITIALIZE(), $\hat{\delta} \leftarrow$ -1
2: $\hat{\mu}_{z_\tau} = \mu_{z_\tau}, \hat{\sigma}_{z_\tau} = \sigma_{z_\tau}, \forall \tau \in \Gamma$
3: while NOTEQUAL($\delta, \hat{\delta}$) do
4:     $\hat{\delta} = \delta$
5:     $\langle \mathcal{V}, \delta \rangle =$ LPASOLVER($\{\hat{\mu}_{z_\tau}\}, \{\hat{\sigma}_{z_\tau}\}$)
6:     for all $\tau \in \Gamma$ do
7:         $\delta_\tau = \frac{\delta_\tau + \hat{\delta}_\tau}{2}$
8:         $\hat{\mu}_{z_\tau} = \mu_{z_\tau} - q1_\tau * \delta_\tau$
9:         $\hat{\sigma}_{z_\tau} = \sigma_{z_\tau} - q2_\tau * \delta_\tau$
```

In Algorithm 2, INITIALIZE() function is used to set the initial values for the δ variables. The initial values are essentially the minimum possible values for existence of a solution. They are generally set to zero, however in severely oversubscribed scenarios, these values are obtained by solving a simple set of linear equations involving durational uncertainties of tasks. In Algorithm 2, we fix the mean ($\hat{\mu}_{z_\tau}$) and deviation ($\hat{\sigma}_{z_\tau}$) variables and find a solution (using LPASOLVER()), i.e. the expected quality and the δ values (that indicate the amount of shedding required). Subsequently, based on the δ values, the mean and deviation variables are updated and a second call to LPASOLVER is made. This process continues until the δ variables are equal (or within a small number ϵ) of the $\hat{\hat{\delta}}$ variables (check performed by NOTEQUAL() function), at which stage the optimal linear policy is achieved. The $\hat{\hat{\delta}}$ variables correspond to the tasks for which uncertainty was not accounted for.

Proposition 1 *Algorithm 2 converges to a linear policy solution that has the highest expected quality.*

Proof. δ and $\hat{\hat{\delta}}$ correspond to tasks that have to be eliminated and the tasks for which uncertainty is not accounted for by the LPASOLVER() respectively. This proposition involves proving two sub-parts:
(a) Convergence: If δ is high, $\hat{\hat{\delta}}$ is low and vice versa. This is because, one uses up the free time obtained by the increase in the value of the other. The values of these variables essentially represent the interval within which the optimal solution (shedding) lies. Since ITERATIVESOLVER performs a binary search over this interval, δ and $\hat{\hat{\delta}}$ will become equal (or at least come within a small value, ϵ of each other) after some number of iterations. Thus, the execution exits the "WHILE" loop on line 3 of Algorithm 2.
(b) Optimality of the solution: At convergence, δ and $\hat{\hat{\delta}}$ are equal. Thus, the interval only contains one element and since the interval should contain the optimal solution, δ represents the optimal shedding required. Furthermore, since each call to LPA-SOLVER() maximizes expected quality over all linear policies, we obtain the optimal (expected) linear policy at convergence.
Hence proved. ■

A deficiency of the LPA algorithm is the linear policy assumption. To address this deficiency we introduce an algorithm that can compute "any degree polynomial" policies in the next section.

4.2 Sample based Learning of Policies, SLP

Similar to LPA, this technique computes dynamically controllable policies for the executable time points. At a higher level, there are two key steps to this technique: (a) Sample the durational uncertainty of tasks and compute the maximum quality solution corresponding to every sample; and (b) Using regression, generalize to policies from the solutions for the specific samples. There are two differences between SLP and LPA. First, SLP can be used to compute higher degree polynomial policies, whereas LPA can be used to compute only linear policies. Second, the objective function of the optimization problem in LPA is maximizing expected quality, while in SLP, it is maximizing the actual solution quality for a specific sample of the durational uncertainty.

Algorithm 3 provides the pseudo-code for SLP. As mentioned earlier, the first key step is to compute the solutions for the samples (lines 1-6). These solutions are computed one uncertain event at a time. We compute dynamically controllable solutions and hence the time for an executable event will only depend on values of uncertain events in its dependency set. Therefore, we initially sort the uncertain events (line 1, SORTOBSEVENTS()) in ascending order of their temporal distances from the starting event. Then, we start with the first uncertain event and compute solutions (for all samples of the uncertain edge), $\mathcal{S}^1$, for executable events (E^1) that are affected (only) by

Algorithm 3 SLP-SOLVER()

```
1: SortObsEvents($\{\tau_e\}$)
2: for all $i \leq |\Gamma|$ do
3:     $E^i \leftarrow$ GetAffectedEvents($1 \cdots i$)
4:     $\mathcal{M}^i \leftarrow$ GetSamples($1 \cdots i$)
5:     for all $j \leq |\mathcal{M}^i|$ do
6:         $\mathcal{S}^i[E^i, \mathcal{M}^i_j] \overset{+}{\leftarrow}$ GetSolSample($\mathcal{S}, \mathcal{M}^i_j$))
7: ConstructSolutionPolicies($\mathcal{S}, degree$)
```

that uncertain event (lines 3-6 of Algorithm 3). GETSAMPLES() returns all the samples associated with a set of uncertain events. After that, we compute solutions ($\mathcal{S}^2$)for executable events (E^2) affected (only) by the first two uncertain events[3] and so on.

Algorithm 4 GETSOLSAMPLE($\mathcal{S}, \mathcal{M}^i_j$)

```
1: max $\sum_{\tau \in \Gamma} f_\tau(X_\tau) + \mathcal{P}(\delta)$
2: variables: $\{\tau_d\}, \{\tau_s\}, \{\delta_\tau\}$
3: s.t.
4: $\forall \tau \in \Gamma$
5: $X_\tau = \mathcal{S}^{|D_\tau|}[\tau_d, \mathcal{M}^i_j] - \mathcal{S}^{|D_\tau|}[\tau_s, \mathcal{M}^i_j]$, if $|D_\tau| < i$
6: $X_\tau = \tau_d - \tau_s$, if $|D_\tau| \geq i$
7: $lb_\tau \leq \tau_d - \tau_s + \tilde{\delta}_\tau \leq ub_\tau$
8: $lb_{\tau,\tilde{\tau}} \leq {\tau'}_s - \{\tau_d + z_\tau\} \leq ub_{\tau,\tilde{\tau}}$, if $|D_\tau| > i$
9: $lb_{\tau,\tilde{\tau}} \leq {\tau'}_s - \{\tau_d + \mathcal{M}^i_j[\tau_e]\} \leq ub_{\tau,\tilde{\tau}}$, if $|D_\tau| \leq i$
10: $\tilde{\delta}_\tau = \delta_\tau + \sum_{\tau' \in \Gamma} g_{\tau',\tau} \delta_{\tau'}$
```

Algorithm 4 defines the optimization problem used for computing the solution for a specific sample (GETSOLSAMPLE()). Solving this optimization problem yields time values for executable events, which are affected by the uncertain events in consideration. In similar vein to LPA, SLP employs slack variables, δ that provide the task compression/shedding values. The objective of this optimization problem is to maximize the sum of the quality obtained from all the tasks and a penalty for reduction in task load, $\mathcal{P}(\delta)$ (line 1). This optimization is constrained by duration constraints, temporal and generate dependencies between tasks (lines 7-10). If τ_e is part of the uncertain events in consideration, the value of z_τ is retrieved from $\mathcal{M}$ (line 9), otherwise it is set to $\mu_{z_\tau} \pm \sigma_{z_\tau}$ (line 8). Similarly, if the solution for some of the executable events was already computed for a sample, then it is retrieved from S (line 5).

The second key aspect of the SLP (Algorithm 3) is that policies are learned from the individual solutions using regression (line 7). The policies are computed one executable event at a time. $\mathcal{M}^{D_\tau}$ denotes the set of samples corresponding to the uncertain events in dependency set, D_τ (for an executable event corresponding to task τ). $\mathcal{S}^{D_\tau}$ specifies the value for the executable event for each sample of the uncertain events in

[3]Since the uncertain events are temporally sorted, no executable event can have only the last r (>0 and $<$ i) of the i events in its dependency set.

its dependency set. Let the polynomial policy matrix be specified by Π. Then,

$$\mathcal{M}^{D_\tau}\Pi = \mathcal{S}^{D_\tau} \Rightarrow \Pi = ((\mathcal{M}^{D_\tau})^T\mathcal{M}^{D_\tau})^{-1}(\mathcal{M}^{D_\tau})^T\mathcal{S}^{D_\tau} \tag{4}$$

Since we already know $\mathcal{M}^{D_\tau}$ and $\mathcal{S}^{D_\tau}$, Π can be solved using equation 4 above. Based on the degree of the polynomial desired, the structure of these matrices changes. A minor issue with this approach is that regression can occasionally have learning errors, i.e. it can generate policies that may not satisfy all the temporal constraints. This can be rectified by making local modifications to the schedule during run-time to fix errors.

Whereas the computational complexity of LPA is polynomial, SLP is exponential in the number of uncertain nodes in the STPPU, primarily because of exponential number of samples. Even though SLP is exponential, because it solves smaller optimization problems and because there are fewer super tasks (in general), the runtime performance is comparable to that of LPA.

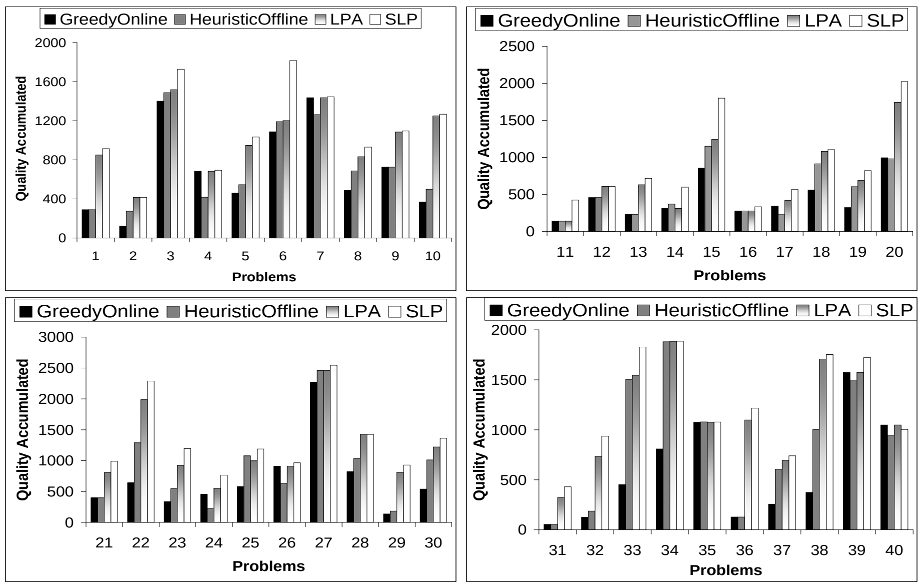

Figure 1: Quality comparison for 'average number of tasks generated' between (a) 0 and 1; (b) 1 and 2 (c) 2 and 3 (d) 3 and 4.

5 Experimental Results

In this section we evaluate the performance of our algorithms in comparison to two baseline deterministic procedures representative of prior approaches to oversubscribed scheduling problems in other domains: an online, reactive procedure and an off-line search-based procedure. We first motivate and summarize these baseline procedures. Next we introduce a run-time re-scheduling procedure to allow off-line scheduling procedures to take advantage of opportunities when actual numbers of generated tasks deviate from expected. We then briefly summarize the set of test problems and experimental design assumptions. Finally, we describe the results of the set of experiments performed.

Benchmark Algorithms: In recent years, there has been a surge of algorithms for solving deterministic oversubscribed scheduling problems. One broad class has focused on solving the combined ordering and shedding problem using off-line search techniques (e.g., [1, 12, 8]). Other work (e.g.,[9]) has emphasized on-line reactive techniques for ordering and dropping tasks as circumstances change. Neither class of techniques is directly applicable to our task management problem, whose features combine uncertainty in both task durations and future task load with oversubscription. At the same time, we can define representative baseline variants for each of these classes to analyze the relative advantage that our STPPU solving techniques can provide over deterministic counterparts.

We define a *greedy online* procedure by modifying the recently developed technique of [9] to incorporate our more general notion of quality profile and incrementally update its schedule as new tasks are generated over time (thus accounting for uncertainty in future task load). To define a *heuristic off-line* procedure, we take a different approach (due to the complexity of augmenting prior techniques to take uncertainty in future task load into account). We define a deterministic version of LPA as our second baseline algorithm, by modifying the procedure to assume expected value for durational uncertainty and consider the duration for the task that yields the highest quality.

One important difference between the *heuristic off-line* procedure just defined and the above mentioned techniques for deterministic oversubscribed scheduling is that the former assumes that a task order is given as input (as does LPA and SLA). For the experiments reported below, we assume that a simple topological sort is used as an ordering heuristic for all three off-line procedures. Note however, that the introduction of a better task ordering will benefit any of these procedures. Heuristic techniques for computing task orderings such as precedence constraint posting [18], squeaky wheel optimization [10] and genetic algorithms [1] can be coupled with our oversubscribed STPPU solving techniques to provide the same leverage that they do in deterministic domains.

Run-time re-scheduler To account for unexpected situations at run-time, we introduce a run-time re-scheduler to operate in conjunction with all three off-line approaches. These deviations arise because of: (a) Deviations from expected load: Fewer or more tasks than the expected set of tasks can be generated at run-time. (b) Learning errors in SLP (as noted in Section 4.2). To handle such situations, the run-time re-scheduler modifies the schedule locally using: (a) Idle waiting: if no task can be scheduled currently, an idle-wait is introduced until there is no constraint violation; or (b) Scheduling shed tasks: If fewer tasks were generated (than the planned number of tasks), then the free time is utilized opportunistically by scheduling the best subset of previously eliminated tasks that do not violate any constraints; or (c) Advancing start time: In (b) above, if no task can be scheduled, then the next scheduled task is moved to an earlier start time without violating any constraints. This will make free time available later.

Experimental Setup: Our work is motivated by a larger project aimed at developing an expert assistant for planning a conference, and our evaluation focuses on instances of the task management problem that arise in this setting. As in most urban office settings, tasks in this domain typically originate from EMails (e.g., ”find a larger room for the keynote talk”) and there are 10 basic types of tasks (corresponding to different aspects of the conference planning problem) with temporal and generate dependencies between them.

For the experiments described below, we generated 40 test problems in which the number of task types were varied from 5-10, and the total number of tasks were varied between 50-200. For all problems, the durations of tasks, quality profiles and durational uncertainty were generated randomly in accordance with domain parameters. Since new tasks are generated at run-time according to a probabilistic model, the quality accumulated for any problem is an average over 25 runs. We ran the four approaches: *greedy online*, *heuristic offline*, LPA and SLP (with 2nd degree polynomial policies) on all problem instances. The run-time re-scheduler was used in conjunction with all the approaches. The results presented in this section are statistically significant with t-test value less than 0.01.

Results: Our first set of results, Figure 1 show the performance comparison between LPA, SLP, *greedy online* and *heuristic off-line* algorithms on the suite of 40 problems. We varied the average number of tasks generated from executing a task from 0-4. Figure 1(a) provides results for problems where the average number of tasks generated was between 0 and 1, in Figure 1(b) the number of tasks generated was between 1 and 2, and so on. Run-time for SLP was in the order of 10-20 seconds, while LPA and *heuristic off-line* were in the order of 1-5 seconds. Greedy took less than 1 second for its online processing. To provide an idea of the oversubscription in these problems, around 10%-60% of the tasks were scheduled. In all four graphs, X-axis denotes the problems and Y-axis denotes the quality obtained at run-time.

These results demonstrate the advantage of producing schedules with an explicit model of uncertainty and preference relative to purely reactive and offline heuristic search based approaches. Both SLP and LPA outperformed the greedy approach in 95% (38/40) of the problems and the overall average quality obtained by SLP and LPA across all problems was approximately 89% and 67% better than greedy respectively. Furthermore, LPA and SLP dominated the *heuristic off-line* approach in 90% (36/40) and 95% (38/40) of the problems respectively. With respect to overall average quality, SLP and LPA obtained 52% and 35% better than the *heuristic off-line* procedure. In the few cases (2/40) where the reactive procedure performed marginally better than SLP and/or LPA (problems 26 and 40), the average number of tasks generated was either 2-3 or 3-4. Since, both SLP and LPA base their solutions on the expected number of tasks, these results may simply reflect the fact that there is higher chance of deviation from expected behavior when more new tasks are generated. With respect to the STPPU techniques, SLP outperformed LPA 97% of the time (39/40). The dominance of SLP over LPA could be attributed to two reasons: (a) second order polynomial policies employed by SLP; and (b) the objective function (actual quality in SLP as opposed to expected quality in LPA).

Our second set of results (Figure 2) illustrate the impact of uncertainty on the performance of the different algorithms. X-axis indicates the percentage of durational uncertainty (relative to the task durations) and Y-axis indicates the quality accumu-

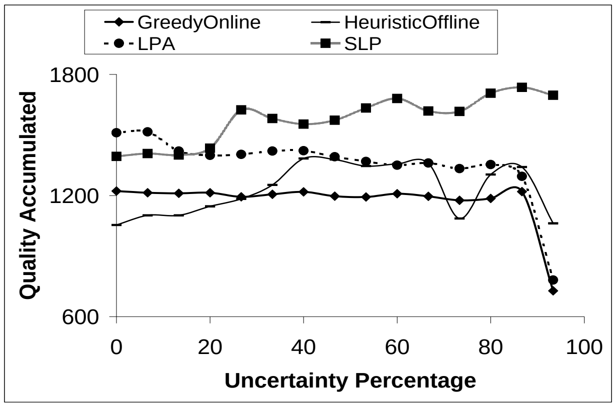

Figure 2: Comparison of solution quality as uncertainty is increased

lated. For this experiment, we varied only the durational uncertainty while keeping the rest of the parameters fixed. There is an interesting trend in the performance of the algorithms. At low levels of durational uncertainty (0%-20%), LPA dominates all the other algorithms and its performance gradually decreases as uncertainty increases. On the other hand, SLP is dominated by LPA for low values of uncertainty and as uncertainty increases (20% - 100%), it dominates all the other algorithms. *Heuristic off-line* procedure has a similar performance graph to LPA, however, at lower levels of uncertainty, it fares poorly.

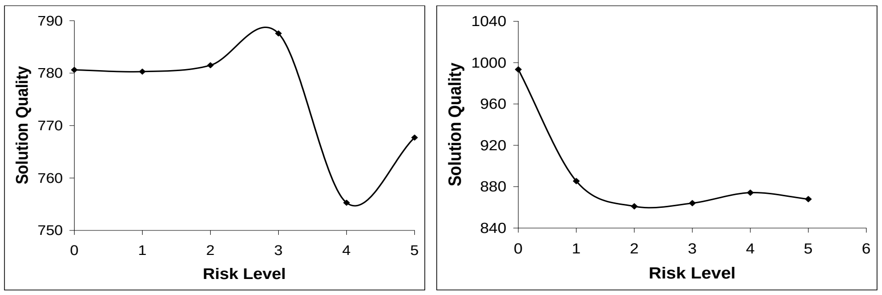

Figure 3: Solution quality vs Risk in (a) LPA; (b): SLP.

Figure 3 shows the results of our third experiment designed to examine the effect on solution quality of taking a more aggressive approach to hedging against durational uncertainty. Both SLP and LPA were modified to operate with a parameterized risk level, which specifies the fraction of the durational uncertainty that is to be planned for. Risk level 0 corresponds to planning for the entire interval of durational uncertainty, while risk level 5 corresponds to not planning for uncertainty at all (and reacting to it at run-time). Although there is no general trend in solution quality as the risk level is increased, it is clearly the case that planning for the entire uncertainty (risk level 0) provides a solution that is better than not planning at all for uncertainty (risk level 5).

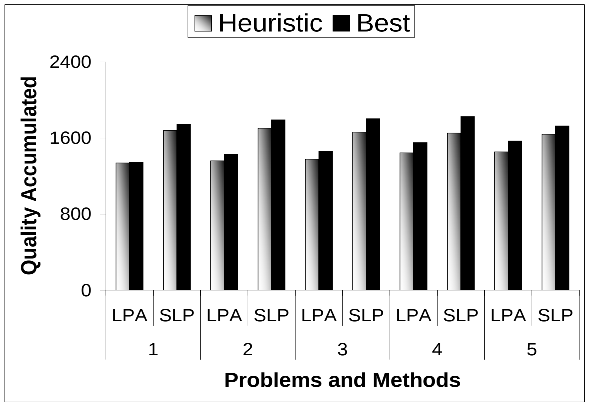

Figure 4: Quality accumulated with SLP and LPA based on the ordering heuristic

Finally, our last set of results show the opportunity for future work in this area. As mentioned earlier, we employ a simple heuristic for introducing ordering dependencies in case of resource conflicts. Figure 4 illustrates the quality difference between the best ordering and the current ordering on some of the smaller problems (where we were able to generate all possible orderings). As expected, we obtained higher solution quality in all the five cases (with both SLP and LPA) with the best ordering heuristic. Investigating the impact of various task ordering heuristics (developed for deterministic problems) in task management domains remains an area for future work.

6 Related Work

There has been substantial research in scheduling under uncertainty, scheduling with preferences/quality and oversubscribed scheduling, but none of this work has considered all three problems simultaneously.

Beck *et al* [2] provide solutions for scheduling with durational uncertainty in job shop scheduling. STNU (Simple Temporal Network with Uncertainty) framework developed by Vidal *et al.* [21] is an extension of the STN (Simple Temporal Network) model by Dechter *et al* [7], models the uncertainty present in task durations. [15] provide a polynomial time solution for computing dynamically controllable solutions to STNU (Simple Temporal Networks with Uncertainty). [13] present techniques for probabilistic controllability of STNU, i.e. allowing for probabilistic violation of constraints. Hence, it provides solutions for some cases where a dynamic controllable solution does not exist. However, it cannot handle problems where there is a need to eliminate tasks (i.e., oversubscribed settings).

STPP (Simple Temporal Problems with Preferences) [11] models scheduling problems with semi-convex utility preference functions over task durations. Khatib *et al* [11] also discuss computation of scalable solutions to STPPs. STPPU (Simple Temporal Problems with Preferences and Uncertainty) [16] is a model that extends STPPs to account for uncertainties. Polynomial time solutions for computation of dynamically controllable solutions to STPPUs were provided by [16].

As discussed earlier, several search procedures have been developed to solve oversubscribed scheduling problems [8, 12, 1] , by making deterministic modeling assumptions. Other work [9] has emphasized online techniques that treat uncertainty reactively.

7 Conclusion

In this paper, we have taken steps toward the development of tools for assisting busy users in managing their workload. Our approach is to model the task management problem as an oversubscribed scheduling problem; and generate solutions that provide explicit advice on which tasks to ignore and de-emphasize if time is short. Given the uncertainty associated with task execution in this domain, we employ a variant of STPPU to model the scheduling problem. Since existing STPPU solution approaches are designed to compute complete solutions and are not applicable in oversubscribed situations, we introduced two new techniques for solving an over-subscribed STPPU. The key idea in these techniques is to introduce slack variables, which get assigned task compression values when no complete solution exists. Experimental results demonstrate the performance advantage of these offline planning techniques (running in conjunction with a run-time re- scheduler) over a purely reactive and a heuristic search approach, which are typical of the usual approaches for handling oversubscribed settings.

Acknowledgements This research was sponsored in part by the Department of Defense Advanced Research Projects Agency (DARPA) un- der contract #NBCHD030010 and by the CMU RObotics Institute.

References

[1] L. Barbulescu, D. L.Whitley, and A. E. Howe. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *AAAI*, 2004.

[2] J. Christopher Beck and Nic Wilson. Proactive algorithms for job shop scheduling with probabilistic durations. *JAIR*, 28:183–232, 2007.

[3] V. Bellotti, N. Ducheneaut, M. Howard, I. Smith, and R. Grinter. Quality versus quantity: Email centric task management and its relation with overload. *HCI*, 20:89–138, 2005.

[4] John Birge and Marc Teboulle. Upper bounds on the expected value of a convex function using gradient and conjugate function information. *Mathematics of Operations Research*, 14(4), 1989.

[5] H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, and M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *IAAI)*, pages 51–58, 2001.

[6] K. Conley and J. Carpenter. Towel: Towards an intelligent to-do list. In *AAAI Spring Symposium*. AAAI, 2007.

[7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *AIJ*, 49:61–95, 1991.

[8] J. Frank, A. Jonsson, R. Morris, and D. Smith. Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space*, 2001.

[9] A.T. Gallagher, T.L. Zimmerman, and S. Smith. Incremental scheduling to maximize quality in a dynamic environment. In *ICAPS*, 2006.

[10] David E. Joslin and David P. Clements. Squeaky wheel optimization. *JAIR*, 10:353–373, 1999.

[11] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *IJCAI*, 2001.

[12] L. Kramer and S.F. Smith. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *IJCAI*, 2003.

[13] H.C. Lau, J. Li, and R. Yap. Robust controllability of temporal constraint networks under uncertainty. In *ICTAI*, 2006.

[14] F. Locatelli, P. Magni, and R. Bellazzi. Using uncertainty management techniques in medical therapy planning: A decision-theoretic approach. In *Applications of Uncertainty Formalisms*, 1998.

[15] P. Morris and N. Muscettola. Temporal dynamic controllability revisited. In *IJCAI*, 2005.

[16] F. Rossi, K. B. Venable, and N. Yorke-Smith. Uncertainty in soft temporal constraint problems: a general framework and controllability algorithms for the fuzzy case. *JAIR*, 27:617–674, 2006.

[17] N. Schurr and M. Tambe. Using multiagent teams to improve the training of incident commanders. In *AAMAS Industry Track*, 2006.

[18] S. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *AAAI*, 1993.

[19] S. Stumpf, X. Bao, A. Dragunov, T. Dietterich, J. Herlocker, K. Johnsrude, L. Li, and J. Shen. The tasktracer system. In *AAAI*, 2005.

[20] Ben Taskar. Learning structured prediction models: A large margin approach. *Stanford University*, 2004.

[21] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI*, 11(1):23–45, 1999.