A Dynamic Programming Formulation of Scheduling Personal Non-Deterministic Activities with Arbitrary Stochastic Durations

Ioannis Refanidis

University of Macedonia, Dept. of Applied Informatics
Egnatia str. 156, 54006, Thessaloniki, Greece

yrefanid@uom.gr

Abstract Scheduling personal activities is a non-deterministic stochastic constraint optimization problem. Activities may have discontinuous temporal domains and arbitrary stochastic durations; they succeed with some probability; they have utilities, whereas several non-trivial constraints and preferences may hold over them, etc. In this article we propose a framework based on dynamic programming, which allows for optimal policies. We also propose a heuristic-based relaxation of the dynamic programming approach, trying to confront the curse of dimensionality. With the relaxed approach we obtain lower bounds of the overall utility. An empirical analysis compares the two approaches in terms of effectiveness and efficiency.

Keywords *Intelligent Calendar Applications, Scheduling, Dynamic Programming, Heuristics*

1 Introduction

In the last decade, many people are using electronic calendars to organize their time. Modern electronic organizers, such as MS-Outlook, Google Calendar and Yahoo Calendar, do not provide for automated scheduling of a user's activities. Users have to manually place their activities in the calendar, as well as to arrange meetings with others. These applications provide various functionalities to assist the user to put her activities into the calendar, detect conflicts, merge calendars, assign tasks to other users, arrange meetings, share and publish the calendar. The need for intelligent assistance to schedule a user's activities has already been identified (Refanidis, McCluskey and Dimopoulos, 2004).

There are several research efforts in the recent years to embed intelligence into calendar applications. Some of them concern meeting scheduling (Berry et. al., 2006; Modi et. al., 2004; Singh, 2003), with personal activities being considered as busy time in a user's plan, whereas others cope only with the problem of scheduling individual activities, without any provision for meeting scheduling (Refanidis and Alexiadis, 2008). Most importantly, all these efforts consider personal activities as deterministic, fully predictable tasks, that is, a form of a closed world assumption. Rescheduling is employed if anything goes wrong.

Although rescheduling is usually an efficient and easy to implement strategy, it does not lead to optimal solutions. On the other hand, finding an optimal policy through dynamic programming is usually computationally prohibitive, whereas it makes strong assumptions such as the knowledge of the probability distributions.

In this article we follow the hard but principled approach: We propose a dynamic programming formulation for the problem of scheduling an individual's activities, characterized by discontinuous temporal domains, arbitrary stochastic durations, ability to interrupt and continue at a later time or not, probability of successful execution, compatible locations, binary constraints and preferences. We implement the proposed dynamic programming formulation for a simple class of problems and we compare it with a family of relaxed versions of it in terms of effectiveness and efficiency. As expected with relaxed dynamic programming approaches (Lincoln and Rantzer, 2006), the results show a significant speedup for the relaxed versions compared to the full dynamic programming approach, as well as a reduction in the utility of the resulting policy especially for tight problems. Our work can also serve as reference to compare with other non-optimal strategies such as stochastic Monte-Carlo simulations.

The rest of the article is structured as follows: First we present the motivation behind this work, having its origins in existing work on scheduling personal activities in electronic calendars. Then, we present background

information on stochastic scheduling. Next, we model the problem of scheduling non-deterministic activities with stochastic durations and, subsequently, we elaborate a dynamic programming approach to solve optimally various classes of this problem. A heuristic-based relaxed dynamic programming approach is also proposed to cope with the curse of dimensionality. Empirical results demonstrate the effectiveness and efficiency of the proposed approaches for the simplest class of problems. In the last section we conclude the article and identify interesting directions for future work.

2 Motivation

There are several types of stochastic personal activities, mainly classified according to their duration profile, interruptibility and determinism. The simplest type is a non-interruptible deterministic activity with stochastic duration. As an example, consider doing transactions at the bank. Depending on the number of people in the queue, not known in advance, the activity's duration varies.

Activities with stochastic durations might exhibit arbitrary distributions. The most common duration distribution is the normal, whereas the uniform distribution is simpler in use. More complex distributions are also possible. As an example, assume a teacher that has to evaluate students' projects. Assuming a normal distribution centered at 30 mins for the time needed to evaluate a single project, as well as a small number of submitted projects not known yet to the teacher, the duration distribution for this activity has peaks at every 30 mins.

An activity's duration profile might change when new information arrives. It is quite common to get a more precise estimate of an activity's duration upon or close to its start. For example, going to the bank or to a doctor without an appointment has a broad a priori duration distribution; however, as long as such an activity starts, the person gets a better estimate of the activity's duration based on the people in the queue. On the other hand, waiting at the street for a taxi has approximately the same expected remaining time, irrelevant to how much time has already passed.

Activities may be interruptible or not. A non-interruptible activity that is stopped before its completion needs to get restarted. For example, waiting in the queue of a bank is a non-interruptible activity; if you decide to leave the queue, you have to reenter the queue at a later time from its beginning. On the other hand, interruptible activities can be interrupted and resumed at a later time from the point of interruption. The teacher that evaluates students' projects is such an example.

A restarted activity does not have to exhibit the same duration in every trial. For example, restarting the activity of waiting in a doctor's office does not have the same a posteriori duration profile (although both trials have the same a priori duration profile). So, experiencing a long expected duration might serve as a reason to interrupt this activity and try to visit the doctor another time. Waiting for a taxi is a case of an activity that does not exhibit the same actual duration upon restart (although, by restarting this activity you won't know the duration of the first trial). On the other hand, there are other activities that exhibit the same duration in all trials. As an example, consider the activity of hiking an unknown trail; if you decide to interrupt this activity and restart at a later time, you have to start hiking from the beginning of the trail. There is no reason to expect that the time needed to hike the trail would be different in the two trials. Of course, any subsequent trial might have an updated duration profile, since a new lower bound for the duration is known.

Finally, an activity might be non-deterministic. This means that after the activity has started, it might fail at any time (even at the end), without yielding any utility. Examples of non-deterministic activities are going to a restaurant without a reservation, giving exams to get a diploma, submitting an article to a journal etc. Failed activities need to be rescheduled.

Scheduling personal activities can be considered a constraint optimization problem (Refanidis and Yorke-Smith, 2010). Each activity has a temporal domain associated with it. Binary constraints, e.g., ordering or proximity, between the activities might exist. Utility might arise not only from accomplishing an activity, but also from the way an activity is scheduled within its temporal domain (unary preferences), as well as from the way activities are jointly scheduled (binary preferences). Hence, leaving an activity out of a plan might result in receiving more utility from the way the remaining activities are scheduled.

3 Background: Stochastic Scheduling

In stochastic scheduling (Righter, 1994) it is assumed that the distributions of the stochastic variables are all known in advance. In what follows, random variables are capitalized, while their actual values (when they get known) appear in lowercase. For example, D_i refers to the stochastic duration of activity T_i, whereas d_i refers to its actual duration.

There are several classes of density functions that can be used to describe distributions (Barlow and Proschan, 1975). Assuming time is discrete, an assumption common in electronic calendars, we are interested in discrete time distributions; however, there are analogous distributions for continuous time variables. An important discrete time distribution is the deterministic distribution, which assigns probability one to a specific value. Another important class is the geometric distribution, which assigns probability $P(X=t) = (1-q)q^t$, whereas $P(X \leq t) = 1-q^{t+1}$ and $E[X] = q/(1-q)$. The uniform distribution assigns probability $1/K$ to all values between $[a..a+K-1]$, for some a and K, and zero to any value outside this interval.

The completion rate of a discrete time random variable is defined as:

$$c(t) = \frac{P(X = t)}{P(X \geq t)} \quad (1)$$

For the geometric distribution, the completion rate is equal to $1-q$, so it is independent of t. This is called the *memoryless* property, which, assuming that X expresses the duration of an activity, implies that the probability to complete the activity at a specific time point (provided it has not been completed before) is always the same. The duration of activity "waiting for a taxi" could be considered as exhibiting the memoryless property.

Distributions can also be classified according to their completion rate. An *Increasing Completion Rate* (ICR) distribution is one for which $c(t)$ is a monotonously increasing function, i.e., as we proceed in time, the probability to complete the activity at any time point increases. On the other hand, a *Decreasing Completion Rate* (DCR) distribution has monotonously decreasing completion rate, i.e., as we proceed in time, the probability to complete the activity decreases. The geometric distribution is both ICR and DCR, since it has a constant completion rate. Of course, there are distributions that are neither ICR nor DCR.

In what follows we are interested in maximizing the expected utility obtained by a particular policy, which is the simplest form of dominance between distributions (Ross, 1995). We are also interested in preemptive dynamic policies, i.e., preemption is allowed for all activities (irrelevant to whether they are interruptible or not), whereas decisions can be taken at any time, based on all the available information.

Scheduling individual stochastic activities resembles stochastic single machine models (Pinedo, 2008). Of particular interest for our problem is the definition of the Gittins index (Gittins, 1979), which is applied to problems with preemptions and arbitrary distributions. For a particular activity T_i, the Gittins index is defined as the largest utility that is obtainable by dividing the total (discounted) expected reward over a given time period (determined by the stopping time) by the (discounted) time itself. So, an optimal policy for single machine problems is to use the Gittins index to determine the activity to be executed and the optimal time to break this activity (provided that it has not been completed before). However, this policy assumes that all jobs are available from the beginning and that they share the same temporal domain. For more complex situations, as the one described in this article, the Gittins index is not directly applicable.

4 The Model

The model presented in this section extends the model presented in (Refanidis and Yorke-Smith, 2010). Time is considered discrete, modeled with non-negative integers, with zero denoting the current time. Consider a set T of N activities, $T=\{T_1, T_2, \ldots, T_N\}$. We suppose that each activity T_i has a stochastic duration D_i following a discrete distribution $p_i(D_i)$. With d_i we denote the actual duration of T_i, $p_i(d_i)>0$. We use the symbols $d_i^{\min}$ and $d_i^{\max}$ to refer to the minimum possible and maximum possible durations of an activity, according to p_i; $d_i^{\max}$ could also be infinite.

An activity T_i may be non-deterministic. The probability of its successful completion is denoted with q_i. We are informed about T_i's successful or unsuccessful execution only at the end of its duration.

An activity T_i may be (but need not be) *interruptible*, i.e., it may be possible to split the activity into parts that can be scheduled separately. With the decision variable $parts_i$ we denote the number of parts into which T_i has been split in a plan ($parts_i \geq 1$; for non-interruptible activities we always have $parts_i=1$). With T_{ij} we denote the j^{th} part of T_i, $1 \leq j \leq parts_i$. We use t_{ij} and d_{ij} to denote the actual start time and duration of T_{ij}, respectively. The sum of the durations of all parts of an activity equals d_i:

$$\forall T_{ij}, \ \sum_{j=1}^{p_i} d_{ij} = d_i \quad (2)$$

Several constraints and preferences may hold over the way interruptible activities are split into parts and the way these parts are scheduled. For example, these constraints and preferences may concern the sizes of these

parts or their temporal distances (minimum and maximum).

Each activity T_i has a temporal domain $Domain_i=[a_{i1},b_{i1}]\cup[a_{i2},b_{i2}] \cup\ldots\cup [a_{iFi},b_{iFi}]$, consisting of a set of intervals, with F_i denoting their number. The temporal domain determines the time periods when the parts of an activity can be scheduled:

$$\forall T_{ij}, \quad \exists k, 1 \le k \le F_i : a_{ik} \le t_{ij} \le b_{ik} - d_{ij} \qquad (3)$$

It is expected that $a_{ij}+smin_i \le b_{ij}$ as well as $b_{ij} < a_{i,j+1}$ for each $1 \le i \le N$, $1 \le j \le F_i$.

A set of M locations, $Loc = \{L_1, L_2, \ldots, L_M\}$ and a two dimensional matrix $Dist$ of the temporal distances between pairs of locations (non-negative integers), not necessarily symmetric, are given. Each activity T_i is associated with a non-empty set of locations, $Loc_i \subseteq Loc$, denoting alternative places where the person should be in order to accomplish each part of the activity. The decision variable $l_{ij} \in Loc_i$ denotes the particular location where T_{ij} is scheduled. For parts of the same or different activities scheduled at distant locations, the following condition should hold:

$$l_{ij} \in Loc_i \qquad (4)$$

$$\forall T_{ij}, T_{mn}, T_{ij} \neq T_{mn} \wedge (Dist(l_{ij},l_{mn})>0 \vee Dist(l_{mn},l_{ij}) > 0) \Rightarrow t_{ij}+d_{ij}+Dist(l_{ij},l_{mn}) \le t_{mn} \vee t_{mn}+d_{mn}+Dist(l_{mn},l_{ij}) \le t_{ij} \qquad (5)$$

Binary constraints, such as ordering, proximity and implication, may hold over the activities. No valid policy should allow scheduling activities in such a way that a binary constraint is violated. With the exception of implication constraints, all other binary constraints are ignored in case one of the involved activities is not scheduled.

Scheduling an individual's activities is considered to be an optimization problem in the proposed model, that is, the goal is not to schedule all the activities, but to maximize the expected utility. There are three aspects of the problem that contribute to the utility: which activities are included in the plan, how each activity is scheduled individually, and how the activities are scheduled in relation to each other.

To summarize, scheduling non-deterministic personal activities with stochastic duration is formulated as follows:

Given:

1 a set of N activities, $T=\{T_1, T_2, \ldots, T_N\}$, each one of them characterized by its utility, a probability of successful execution, a stochastic duration distribution, a temporal domain and a temporal domain preference function, a set of alternative locations, the interruptibility property with related constraints and preferences
2 a two-dimensional matrix with temporal distances between all locations
3 a set C of binary constraints
4 a set P of binary preferences

Schedule:
find a policy that maximizes the expected utility resulting from all sources of utility, without violating any of the problem's constraints.

5 A Dynamic Programming Formulation

This section presents a dynamic programming formulation for the problem of scheduling personal non-deterministic activities with stochastic durations. The approach is elaborated gradually, with the simplest case comprising simple non-deterministic activities with arbitrary stochastic duration distributions and the most complex case comprising also unary and binary preferences, binary constraints and interruptible activities.

5.1 Simple Utilities with Non-Interruptible Activities

The simplest case concerns non-interruptible activities, the successful execution of each one of them results in a fixed amount of utility. No binary constraints or preferences are assumed, apart from the requirement that no two activities can be executed in parallel. Each time a decision has to be taken, we have the options either to start executing an executable activity up to a break point, or wait in order to execute another activity at a later time point.

In order for an activity T_i to be executable at a time point t, t should be within its temporal domain, whereas there should be at least $d_i^{\min}$ duration left within the current interval of the activity's temporal domain such that the activity has non-zero probability of successful execution. A second requirement for T_i's executability is that there is a minimum temporal distance of at least $Dist(l_m,l_i)$ between the location l_i selected for the next activity T_i and the location l_m of the previously executed activity T_m. The alternative is to decide not to execute any activity at the current time t, in order to execute an (non-executable at the current time) activity T_j at a later time t'. Activity T_j might be not executable at the current time either because of its temporal domain or because of its location. However, in the first case there is no reason to remain idle if it is possible to execute another activity T_i in between, whereas T_j remains still executable at t'.

$$U(T,t,l) = \max_{\substack{T_i \in T, l_i \in Loc_i \\ d_i^{\min} \le b \le d_i^{\max} \\ \text{such that} \\ t_i = next(T_i,t,l,l_i), t_i < \infty \\ t_i + j \in Domain_i \ \forall j, 0 \le j \le b \\ \neg\exists T_j \in T, l_j \in Loc_j \text{ s.t. } next(T_j,t,l,l_j) < next(T_i,t,l,l_i) \\ \text{and } next(T_i, t_j + d_j^{\min}, l_j, l_i) = next(T_i,t,l,l_i)}} \left\{ \begin{array}{l} \sum_{d_i=d_i^{\min}}^{b} P(d_i) \cdot \left(\begin{array}{l} q_i \cdot (U_i + U(T/T_i, t_i + d_i, l_i)) \\ + (1-q_i) \cdot U(T, t_i + d_i, l_i) \end{array} \right) \\ + P(d_i > b) \cdot U(T, t_i + b, l_i) \end{array} \right\} \quad (6)$$

So, the dynamic programming formulation of the problem is given by Eq. (6), with $U(T,t,l)$ denoting the expected utility by scheduling the activities in T, when the current time is t and the current location is l. In Eq. (6), maximization is performed over T_i, l_i and b, i.e., at each time point t one has to decide which activity to execute (T_i), at which location (l_i) and for how much time at most (b). The earliest possible start time of each activity T_i for a specific location l_i is specified by function *next*, which takes into account $Domain_i$, the current time t and the current location l. In case of a non-executable activity T_i at specific location l_i, $next(T_i,t,l,l_i)$ returns infinite, and this option (i.e., scheduling T_i at location l_i) is ignored by the maximization procedure.

The last two lines of the max conditions are for optimization reasons: They impose constraints to ignore any activity T_i scheduled at location l_i, such that there is another activity T_j that could be scheduled before T_i at time t_j and location l_j for $b_j \geq d_j^{\min}$, such that $next(T_i,t,l,l_i) = next(T_i,t+b_j,l_j,l_i)$. That is, by executing T_j before T_i, there is no change in the time when T_i can be scheduled at l_i, so there is no reason to remain idle.

5.2 Temporal Preferences

We assume that for each activity T_i there is a preference function over the way the activity is scheduled within its domain. For example, one might prefer to schedule the activity as early as possible or as late as possible (monotone functions), or in the morning of any day (non-monotone function). We can define a function $u_i^{time}(t)$ representing the preference for the various time slots of an activity's temporal domain, so the overall temporal utility can be defined as:

$$U_i^{time}(\pi_i) = \frac{\sum_{j=1}^{parts_i} \sum_{t=t_{ij}}^{t_{ij}+d_{ij}} u_i^{time}(t)}{d_i} \quad (7)$$

where π_i is the actual plan of T_i, $\pi_i = \{\langle t_{i1}, d_{i1}, l_{i1}\rangle, \langle t_{i2}, d_{i2}, l_{i2}\rangle, \ldots, \langle t_{i,parts_i}, d_{i,parts_i}, l_{i,parts_i}\rangle\}$. Eq. (7) is general covering also interruptible activities. Especially for non-interruptible activities (i.e., $parts_i = 1$), $U_i^{time}(\pi_i)$ can be considered a function of t_i and d_i, i.e., $U_i^{time}(\pi_i) = U_i^{time}(t_i, d_i)$.

Generalizing Eq. (6) for temporal utilities is not straight-forward. Particularly, a person may remain idle in order to schedule the activities in more desirable time periods, with the risk of failing to execute them successfully within their temporal domains. So, at each time point, one has to take four decisions: Which activity to schedule first, where to schedule it, when to schedule it (not necessarily immediately) and when to break its execution. For any candidate activity T_i, there is a time point t_i^{best}, such that, if we had only T_i to schedule, scheduling it at t_i^{best} would result in maximizing the expected utility resulting from that activity. So, taking into account the other activities as well, there is no reason to remain idle beyond t_i^{best}, provided that T_i is the first activity to schedule and it can be started at t_i^{best} (that is, there are no other constraints, such as distance constraints, prohibiting T_i's execution at t_i^{best}). In general, for any activity T_i being the first to be scheduled, all possible start times between the current time and t_i^{best} should be considered. Eq. (8) generalizes Eq. (6), in order to take into account temporal preferences.

Eq. (8) mainly differs from Eq. (6) in that it maximizes also over the start time t_i of T_i, where t_i ranges from the next possible time $next(T_i,t,l,l_i)$ up to the best time t_i^{best}. Note that in case there is another activity T_j, for which t_j^{best} is much earlier than t_i^{best}, so as T_j can potentially complete its execution before T_i's start at t_i (including travelling time), there is no reason to remain idle till t_i, that is, T_i should not be the first activity to schedule.

Depending on the form of the temporal preference functions, some simplifications are possible. Assuming that $u_i^{time}(t)$ is a monotonously decreasing function, i.e., the earlier T_i is scheduled the better, t_i^{best} is always equal to $next(T_i,t,l,l_i)$. On the other hand, the worst situation is when $u_i^{time}(t)$ is monotonously increasing, i.e., the latest the better; in this case all the possible start times

$$U(T,t,l) = \max_{\substack{T_i \in T, l_i \in Loc_i \\ d_i^{\min} \le b \le d_i^{\max} \\ t_i = next(T_i,t,l,l_i) \vee next(T_i,t,l,l_i) < t_i \le t_i^{best} \\ \text{such that} \\ t_i + j \in Domain_i \ \forall j, 0 \le j \le b \\ \neg \exists T_j \in T, l_j \in Loc_j \text{ s.t. } t_j^{best} + d_j^{\min} + Dist(l_j,l_i) \le t_i}} \left\{ \begin{array}{l} \sum_{d_i = d_i^{\min}}^{b} P(d_i) \cdot \left(\begin{array}{l} q_i \cdot \left(U_i + U_i^{time}(t_i,d_i) + U(T/T_i, t_i + d_i, l_i) \right) \\ + (1-q_i) \cdot U(T, t_i + d_i, l_i) \end{array} \right) \\ + P(d_i > b) \cdot U(T, t_i + b, l_i) \end{array} \right\} \quad (8)$$

$$U(T,t,f(\pi_t)) = \max_{\substack{T_i \in T, l_i \in Loc_i \\ d_i^{\min} \le b \le d_i^{\max} \\ t_i = next(T_i,t,l,l_i) \vee next(T_i,t,l,l_i) < t_i \le t_i^{best} \\ \text{such that} \\ t_i + j \in Domain_i \ \forall j, 0 \le j \le b \\ \neg \exists T_j \in T, l_j \in Loc_j \text{ s.t. } t_j^{best} + d_j^{\min} + Dist(l_j,l_i) \le t_i \\ propagate(\pi_t \cup \langle T_i, t_i, l_i, d_i \rangle)}} \left\{ \begin{array}{l} \sum_{d_i = d_i^{\min}}^{b} P(d_i) \cdot \left(\begin{array}{l} q_i \cdot \left(\begin{array}{l} U_i + U_i^{time}(t_i,d_i) + \\ U_{binary}(\pi_t \cup \langle T_i, t_i, l_i, d_i \rangle) - U_{binary}(\pi_t) \\ U(T/T_i, t_i + d_i, f(\pi_t \cup \langle T_i, t_i, l_i, d_i \rangle)) \end{array} \right) + (1-q_i) \cdot U(T, t_i + d_i, f(\pi_t)) \end{array} \right) \\ + P(d_i > b) \cdot U(T, t_i + b, f(\pi_t)) \end{array} \right\} \quad (9)$$

between $next(T_i,t,l,l_i)$ and the latest possible start time need to be considered.

Another simplification in the assessment of Eq. (8) concerns the case where all activities can be safely executed at their best times. In this case, maximization over t_i, for $t_i < t_i^{best}$, is not needed. The conditions for this simplification to take place are: T_i should have a finite $d_i^{\max}$ or no subsequent activity should exist. In case there are subsequent activities, a sufficient condition is that all remaining activities are deterministic and for any two activities (including T_i) T_j and T'_j, $t_j^{best} + d_j^{\max} \le t'^{best}_j$ or $t'^{best}_j + d'^{\max}_j \le t_j^{best}$ should hold. This also implies that all remaining activities have finite maximum durations, a condition that is not required only for the activity T_j with the maximum t_j^{best}.

5.3 Constraints and Complex Preferences

Binary constraints may affect the possible ways to schedule the remaining activities, based on the already scheduled activities. Similarly, binary preferences may affect our preference on the possible ways to schedule the remaining activities, based again on the already scheduled activities. So, the utility of the current state is not a memoryless function anymore; it depends on the current plan π_t, i.e., what has happened in the past, including unsuccessful attempts to execute some activities. So, we define the expected utility of the current state as a function with three arguments, that is, $U(T,t,f(\pi_t))$, where T is the set of activities not accomplished yet, t is the current time and π_t is the already executed plan (Eq. (9)).

Function f is a feature extraction function over the already executed plan. In order to evaluate binary constraints and preferences that involve both accomplished and remaining activities, we do not need every detail of the already executed plan. Function f extracts exactly those features from the current plan that are necessary for this evaluation. For example, suppose there is a minimum distance constraint between an already executed activity T_i and a pending activity T_j. In order to evaluate this constraint we only need the end time of T_i. Furthermore, in case the imposed minimum distance is less than the distance between T_i's end time and the current time, the distance constraint is trivially satisfied. As an additional example, consider the current location l that was used in Eqs. (6) and (8). This can also be seen as a feature extracted from the current plan, i.e., the location of the last accomplished activity. Generally, feature extraction alleviates the combinatorial explosion problem, since the space of the features extracted from the various plans is usually much more compact than the space of the possible plans.

A plan π can be considered a set of quadruplets of the form $\langle T_i,t_i,l_i,d_i \rangle$, denoting that activity T_i was executed at time t_i, at location l_i with duration d_i (π_i defined earlier in

$$U(T,t,f(\pi_t)) = \max_{\substack{T_i \in T, l_{ij} \in Loc_i \\ smin_i \le b \le smax_i, d_i^0 + b \le d_i^{\max} \\ t_{ij} = next(T_i,t,l,l_{ij}) \vee next(T_i,t,l,l_{ij}) < t_{ij} \le t_i^{best} \\ \text{such that} \\ t_{ij} + x \in Domain_j \ \forall j, 0 \le x \le b \\ \neg \exists T_k \in T, l_k \in Loc_k \text{ s.t. } t_k^{best} + d_k^{\min} + Dist(l_k, l_{ij}) \le t_{ij} \\ propagate\left(\pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, d_{ij} \right\rangle\right)}}$$

$$\left\{ \begin{array}{l} \text{CASE A}: d_i^0 + b \ge d_i^{\min}: \\ \left\{ \begin{array}{l} \sum_{d_i = d_i^{\min}}^{d_i^0 + b} \left(P(d_i \mid d_i > d_i^0) \cdot \left(\begin{array}{l} q_i \cdot \left(\begin{array}{l} U_i + U_i^{time}(\left(\pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, d_i - d_i^0 \right\rangle\right) : T_i) + \\ U_{binary}(\pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, d_i - d_i^0 \right\rangle) - U_{binary}(\pi_t) + \\ U(T / T_i, t_{ij} + d_i - d_i^0, f\left(\pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, d_i - d_i^0 \right\rangle\right)) \end{array} \right) \\ + (1 - q_i) \cdot U(T, t_{ij} + d_i, f(\pi_t / \pi_{t,i})) \end{array} \right) \right) \\ + \left\{ \begin{array}{l} \text{if } T_i \text{ not interruptible}: \\ P(d_i > b) \cdot U(T, t_{ij} + b, \pi_t) \\ \text{if } T_i \text{ interruptible}: \\ P(d_i > d_i^0 + b \mid d_i > d_i^0) \cdot U(T, t_{ij} + b, \pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, b \right\rangle) \end{array} \right. \end{array} \right\} \\ \text{CASE B}: d_i^0 + b < d_i^{\min}: U\left(T, t_{ij} + b, f\left(\pi_t \cup \left\langle T_i, t_{ij}, l_{ij}, b \right\rangle\right)\right) \end{array} \right. \quad (11)$$

this section refers to the subset of π concerning only T_i; in this case, T_i is omitted for brevity). Function U_{binary} evaluates binary preferences according to a specific plan. In its simplest form, U_{binary} would evaluate only those binary preferences, for which the involved activities have already been scheduled. So, the difference:

$$U_{binary}(\pi_t \cup \langle T_i, t_i, l_i, d_i \rangle) - U_{binary}(\pi_t) \quad (10)$$

in Eq. (9) represents the increase (if any) of the overall utility received from binary preferences involving T_i, with the other activity being already scheduled. More elaborated implementations of U_{binary} could project binary preference evaluations before both the involved activities have already scheduled. For example, a binary preference of the form $T_j \Rightarrow T_i$, interpreted as a preference to accomplish T_i each time T_j is accomplished, can be considered satisfied as long as T_i has been accomplished, irrelevant to whether T_j will be accomplished as well or not.

The definition of t_i^{best} needs also some revision. Taking into account the binary preferences, the best time to schedule a single activity is now a function of other activities as well. In case these other activities have already been scheduled, t_i^{best} can be easily computed; otherwise, our ability to compute t_i^{best} depends on the type of the binary preferences. If this is not possible, t_i^{best} should be considered equal to the latest possible time to execute T_i.

Finally, Eq. (9) assumes a function *propagate* over incomplete plans; *propagate* returns true if constraint propagation starting from the current incomplete plan does not fail, otherwise it returns false. Note that failing to propagate constraints does not result in inability to compute utilities; most binary constraints are considered satisfied in case any of the involved activities is not accomplished. However, if no plan, including the empty plan, satisfies the constraints, $U(T,t,f(\pi_t))=-\infty$ is assumed.

5.4 Interruptible Activities

In the presence of interruptible activities, decisions might consider scheduling a part of an interruptible activity, whereas the activity may remain still unaccomplished. The current plan conveys information about how much duration has already been scheduled. Furthermore, another complication arises from the fact that, for a non-deterministic activity T_i, that is $q_i<1$, one is informed about its successful execution only upon the end of T_i's duration.

Each time a part T_{ij} of an interruptible activity T_i is considered, Eq. (11) distinguishes two cases: T_{ij} can be (but need not be) the last part of T_i (Case A) or not (Case B).

In Eq. (11), d_i^0 refers to the already scheduled duration of T_i (being 0 for non-

interruptible activities), whereas d_i refers to the actual total duration of T_i (not known yet). We also assume a minimum and a maximum possible duration for T_{ij}, denoted with $smin_i$ and $smax_i$ respectively. In order for a part to be potentially the last one, $d_i^0 + b$ should be at least $d_i^{\min}$, otherwise another part is necessary (Case B). However, even if $d_i^0 + b$ exceeds $d_i^{\min}$, it might be the case that $d_i > d_i^0 + b$ (provided that $d_i^0 + b < d_i^{\max}$), so another part might be necessary.

Suppose now that $d_i^0 + b \geq d_i^{\min}$, so there are chances for the activity to accomplish with the current part. Eq. (11) considers the probability that the actual duration d_i of the current activity is between $d_i^{\min}$ and $d_i^0 + b$ and the activity succeeds (with probability q_i). However, in Case A there are two subcases for an interruptible activity to fail: Either $d_i > d_i^0 + b$, or the activity fails due to its non-deterministic nature. Non-interruptible activities are also covered by Case A.

6 Relaxed Dynamic Programming

The dynamic programming approach presented in the previous section exhibits a high complexity. In the simplest case (simple utilities with non-interruptible activities), assuming N activities, K time steps and L locations, we have to compute $U(T,t,L)$ for $O(2^N KL)$ cases, i.e., all subsets of N for every time point and every possible location. On the other hand, in the most complex case (binary constraints, complex preferences and interruptible activities), $U(T,t,f(\pi_t))$ has to be computed for $O(N^K 2^N KL)$ times. Indeed, in the worst case $f(\pi_t) = \pi_t$, that is, no feature extraction takes place. The possible plans are roughly K^N, that is, in every past time point one of the N activities could have been executed, whereas all the activities could be remaining (either because they are interruptible or because they are non-deterministic).

In order to accelerate the computation of the utilities, at the cost of their accuracy, we could perform some relaxations. In this article we propose a simple and easily applicable relaxation, called $Best^k$: In order to compute $U(T,t,\pi(t))$, do not maximize over all applicable activities in T but only over the k best of them, while also considering remaining idle till the next time point. For larger values of k we get more accurate estimates. If k equals to N, that is, the number of activities, then full dynamic programming computation is applied.

In order to define the k most promising activities for a time point t, a heuristic is needed. For each activity T_i applicable at time t, we define its weight $w_i(t)$ as follows:

$$w_i(t) = \frac{U_i \cdot q_i}{E[D_i] \cdot Remainder(D_i)} \qquad (12)$$

where $E[D_i]$ is the expected duration of T_i (or the expected remaining duration of T_i for interruptible activities) and $Remainder(D_i)$ is the net size of $Domain_i$ from t and onwards. Eq. (12) favors activities with high utility and success ratio, small expected durations and small remainder domains. So, for each time point t, activities are ordered in decreasing order of their weights, and the k first of them are considered for execution.

7 Empirical Evaluation

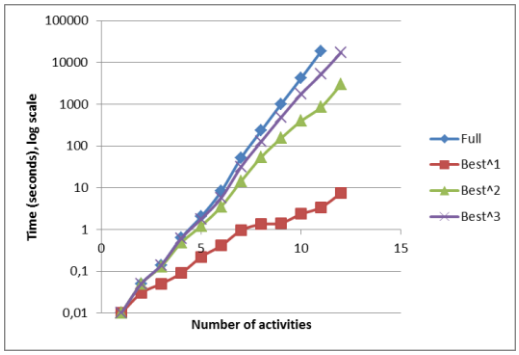
We implemented the proposed algorithms for the simple case of non-deterministic, non-interruptible activities with arbitrary duration distributions. Particularly, we implemented both the complete dynamic programming algorithm as well as the relaxed one. We used ECLiPSe Prolog as our implementation platform. We randomly created a set of 12 activities, $T=\{T_1,T_2,\ldots,T_{12}\}$, with utilities ranging between 5 and 12. One third of the activities are non-deterministic, with probability of successful execution equal to ½. The duration of each activity T_i is a random variable following a random distribution, with $d_i^{\min}$ randomly selected from the interval [2..8] and $d_i^{\max}$ exceeding $d_i^{\min}$ by a random value in [2..6]. Each activity has a temporal domain consisting of two intervals randomly selected within [0..80], with each interval having a width randomly selected from [20..30].

We created 12 problems (#1, #2, …, #12), with problem #i comprising the activities T_1 to T_i. We attempted to solve each problem using both full dynamic programming, as well as the $Best^k$ relaxed version for k=1, 2 and 3. For each attempt we report the time needed to extract the policy, as well as its quality. Table 1 and Fig. 1 present the results.

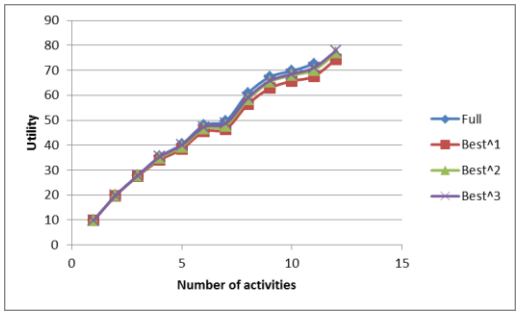
Full dynamic programming gives the actual utility of the best policy, whereas relaxed dynamic programming results in underestimates. However, $Best^k$ produces better estimates of the utility for larger values of k. On the other hand, for smaller values of k we get faster but less accurate results, with no exception. Figure 2 presents the ratio between the utility of the policies obtained by the relaxed versions and the utility of the optimal policy. As can be seen, this ratio generally decreases when problems get harder, that is, more activities within the same time period.

Table 1. Time needed to compute the policies and the corresponding utilities for problems with various numbers of activities using *Best*^{k} relaxation, compared to full dynamic programming. Time is in seconds. Full DP was unable to solve one problem due to time limitations.

#	*Best*^{1}		*Best*^{2}		*Best*^{3}		Full	
	Time	Utility	Time	Utility	Time	Utility	Time	Utility
1	0,01	9,85	0,01	9,85	0,01	9,85	0,01	9,85
2	0,03	19,74	0,05	19,80	0,05	19,80	0,05	19,80
3	0,05	27,54	0,13	27,74	0,14	27,80	0,14	27,80
4	0,09	34,04	0,48	34,86	0,61	35,32	0,63	35,39
5	0,22	38,40	1,22	39,37	1,78	40,04	2,03	40,31
6	0,41	45,46	3,53	46,59	5,7	47,27	8,22	47,99
7	0,95	46,45	14,34	47,7	31,8	48,7	52	49,7
8	1,33	56,39	53,67	58,24	126,72	58,99	231	60,77
9	1,42	62,83	153,64	65,33	485,73	65,82	1006	67,35
10	2,34	65,69	395,92	67,99	1745	68,51	4200	69,72
11	3,38	67,62	852,86	70,14	5218,6	71,10	18441	72,53
12	7,49	74,44	2967,5	76,97	17428	77,88	>50000	-



(a)



(b)

Figure 1. Graphical depiction of Table 1 data. (a) Time to compute the policy. (b) Expected utility for the found policy.

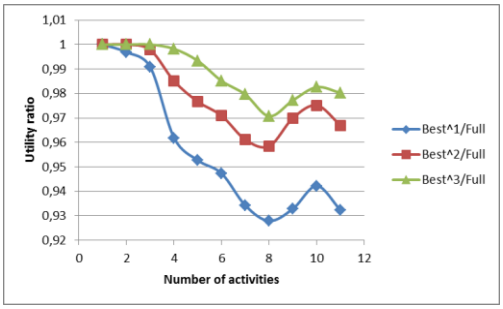


Figure 2. Best policy approximation using the relaxed versions.

8 Conclusions And Future Work

This article formulated the problem of scheduling non-deterministic activities with stochastic durations, discontinuous temporal domains, interruptible or non-interruptible execution, temporal utilities, binary constraints and preferences, using dynamic programming. The motivation behind our work is to provide automated scheduling capabilities to electronic calendars, however the work is easily adaptable to other settings as well. We elaborated a dynamic programming solution for several classes of the problem, ranging from simple non-interruptible activities without binary constraints and preferences to full featured activities. We also proposed a family of heuristic-based relaxations to the proposed dynamic programming solution, in order to compensate the search time with the utility of the resulting policy. Finally, we implemented both the full dynamic programming algorithm, as well as the relaxed version, for the simplest class of problems and we ran several experiments to demonstrate the effectiveness and the efficiency of the various approaches.

For the future we plan to extend our implementation to more rich formulations, that is, temporal preferences as well as binary constraints and preferences, and perform similar experiments. Furthermore, we intend to attempt alternative relaxations, as well as stochastic approaches such as Monte Carlo simulations. Integrating these algorithms in a real calendar application is our ultimate goal.

9 References

Barlow, R.E. and Proschan, F. Statistical Theory of Reliability and Life Testing: Probability Models, Holt, Rinehart and Winston, Inc., New York, 1975.

Berry, P., Conley, K., Gervasio, M., Peintner, B., Uribe T. and Yorke-Smith, N. Deploying a Personalized Time Management Agent, 5th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-06) Industrial Track, Hakodate, Japan, pp. 1564-1571, May 2006.

Gittins, J.C. Bandit Processes and Dynamic Allocation Indices", Journal of the Royal Statistical Society Series B, Vol. 14, pp. 148–177, 1979.

Lincoln B. and Rantzer, A. Relaxing Dynamic Programming. IEEE Transactions on Automatic Control, 51:8, pp. 1249--1260, August 2006.

Modi, P.J., Veloso, M., Smith, S.F. and Oh, J. CMRadar: A Personal Assisstant Agent for Calendar Management, Workshop on Agent Oriented Information Systems (AOIS), New York, 2004.

Pinedo, M.L. Scheduling - Theory, Algorithms and Systems (3rd edition). Springer, 2008

Refanidis I. and Yorke-Smith, N. 2010. A Constraint Based Programming Approach to Scheduling an Individual's Activities. ACM Transactions on Intelligent Systems and Technologies, vol. 1 (2), pp.12:1-32.

Ioannis Refanidis and Anastasios Alexiadis. Deployment and Evaluation of SelfPlanner, an Automated Individual Task Management System. Computational Intelligence, 2011.

Refanidis, I., McCluskey, T.L. and Dimopoulos, Y. Planning Services for Individuals: A New Challenge for the Planning Community, Workshop on Connecting Planning Theory with Practice, Whistler, British Columbia, Canada, 2004.

Righter, R. Stochastic Scheduling. Chapter 13 in Stochastic Orders, M. Shaked and G. Shanthikumar (eds.), Academic Press, San Diego, 1994.

Singh, R. RCal: An Autonomous Agent for Intelligent Distributed Meeting Scheduling, master's thesis, tech. report CMU-RI-TR-03-46, Robotics Institute, Carnegie Mellon University, December, 2003.