Manuscript Click here to download Manuscript mplans.tex

International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

ALTERNATIVE PLAN GENERATION AND ONLINE PREFERENCE LEARNING IN SCHEDULING INDIVIDUAL ACTIVITIES

Anastasios Alexiadis

Department of Applied Informatics, University of Macedonia, Egnatia 156, 54006, Thessaloniki, Greece.
talex@java.uom.gr

Ioannis Refanidis

Department of Applied Informatics, University of Macedonia, Egnatia 156, 54006, Thessaloniki, Greece.
yrefanid@uom.gr

Received (10 October 2014)
Revised (3 August 2015)
Revised (17 December 2015)
Accepted (Day Month Year)

This article tackles a significant aspect of the problem of scheduling personal individual activities, that is, the generation of qualitative, significantly different alternative plans. Solving this problem is important for intelligent calendar applications, since average users cannot adequately express their preferences over the way their activities should be scheduled in time, thus it is common that they are not satisfied by the plans generated for them by a scheduler, although they are near-optimal according to their stated preferences. Hence generating alternative plans and asking the user to select one among them is a sensible approach, provided that the alternative plans are both near-optimal, according to the user-defined preferences, as well as significantly different to each other, in order to increase the chances that at least one of them satisfies the user. Furthermore, based on the assumption that a user might systematically misweight his preferences over the various aspects of a plan, an online non-intrusive method to learn his actual preferences is presented, based on monitoring his selections over the alternative plans. The proposed methods have been evaluated successfully on a variety of problems. Furthermore, they have been implemented in two deployed systems.

Keywords: intelligent calendar applications; scheduling; learning.

1. Introduction

SelfPlanner[1] is a prototype application designed for the automatic scheduling of a user's individual activities in an electronic calendar. It supports a rich domain model for scheduling interruptible/non-interruptible activities, as well as periodic activities with various types of constraints and preferences that can be defined on them by the user.

In this article we extend our previous work on scheduling individual activities by introducing new methods to generate qualitative, significantly different alternative plans. In this way we increase the chances that at least one of them meets the user's preferences. According to Kambhampati,[2] in many real world planning scenarios the user's preferences are either unknown or at best partially specified. This renders increasingly complex to capture the user's preferences in a personal activities scheduling problem. Some systems attempt to elicit user preferences in a non-intrusive manner, by presenting alternative plans to the user and building a preference model based on his choices.[3,4,5,6]

Intelligent assistance for time and task management has been targeted by previous AI research.[4,7,8,1,3,9] In common electronic calendar applications, schedules comprise sets of fully specified and independent events. An event is characterized by a fixed start-time, a duration, and optionally a location. Furthermore, many systems support tasks. These represent individual commitments potentially having a deadline to be met (e.g., writing an article or doing homework). Tasks are usually kept in separate task lists and do not have a specific start time. Converting a task to an event is usually a straightforward procedure that is accomplished by dropping the task into the electronic calendar.

SelfPlanner is based on a domain model[10] that treats events and tasks in a uniform way. They are called *activities* and are characterized by a number of attributes, such as a temporal domain, a duration range, a set of alternative locations, as well as unary constraints and preferences over the ways the parts of an interruptible activity are scheduled in time. In addition, the model also supports constraints and preferences (ordering, proximity and implication) over pairs of activities. The model specifies a Constraint Optimization Problem (COP), where the best solution is the one that maximizes the total utility of the plan, linearly aggregating the various sources of utility. In the same work[10] a scheduler, which is based on the Squeaky Wheel Optimization (SWO) framework and is combined with domain-dependent heuristics to automatically schedule activities, is also presented. The scheduler was later extended with a post-processing module that further increases the solution quality via local-search post-optimization.[11,12]

In the literature one can find two general approaches to generate alternative plans with a planner or scheduler. The first approach is based on managing either the set of possible solutions or the order in which they are evaluated by the search algorithm.[13,14] The second one—which we based our alternative plan generation method on—is based on modifying the heuristic of the planning algorithm.[15,16,17]

This article presents our approach to generate qualitative, significantly different alternative plans for the problem of scheduling an individual's activities. The approach is based on the enhancement of the plan evaluation function used by the scheduler during the optimization process with additional attributes that measure the difference to the already generated plans. We define a plan difference function that utilizes domain-specific traits of the problem formulation. In addition, the article introduces a machine learning method to forecast and compensate systematic

errors performed by the user when defining his preferences over the various aspects of a plan.

Producing qualitative, significantly different plans increases the likelihood that one of them satisfies the user. Indeed, suppose that the user is not satisfied by the optimal plan. Making a small change to it, e.g., moving a single activity by a unit of time or decreasing the total duration of a single activity by a unit of time, results to a new plan that is different from the optimal (although quite similar) and has a utility very close to the optimal utility. The user who was not satisfied by the optimal plan, is quite probable not to be satisfied also by these very similar to the optimal plans.

There are many reasons for this situation to happen. The first reason has to do with the user having incorrectly described her preferences in the problem instance. This is quite a common situation, and we tried to overcome it using machine learning, as it is described in the article. Another reason has to do with the limited expressivity of SelfPlanner's model. There might exist user preferences that cannot be expressed in the supported model, so they are not taken into account by the scheduling engine. In both these situations, returning significantly different plans than the already found results in increased chances that the user will be satisfied by one of them.

The article extends previous work of the authors[18] by introducing machine learning in the generation of alternative plans; by presenting fresh experimental results in all cases; by adopting a system oriented view, than a pure algorithmic one; and, finally, by presenting a more extensive related work and discussion of the results.

The rest of this article is structured as follows. Section 2 formulates the optimization problem and illustrates the SWO-based approach. It also gives an overview of the SelfPlanner prototype system. Section 3 focuses on the hybrid evaluation function that considers (among other criteria) differences to already generated plans. This is based on the introduction of a plan difference function to measure the distance between pairs of plans. Section 4 employs machine learning to reveal and compensate systematic errors in expressing a user's preferences, based on his choices over the alternative plans presented to him. Section 5 presents the implementation of all this new technology in the SelfPlanner prototype system. Section 6 demonstrates multiple plan generation on a number of randomly generated problem instances. In addition, it simulates two virtual users' preferences and evaluates the machine learning method. Finally, Section 7 concludes the article and identifies potential directions for future work.

2. Background

In this section we give an overview of the SelfPlanner prototype application. First of all, we formulate the underlying SelfPlanner's Constraint Optimization (COP) problem. Next, we give an overview of the algorithms SelfPlanner uses to solve instances of the problem. Finally, we give an overview of the deployed

SelfPlanner application.

2.1. *Problem Formulation*

In the domain model presented in previous work,[10] time is considered a non-negative integer, with 0 denoting the current time. A set T of N activities, $T = \{T_1, T_2, \ldots, T_N\}$, is given. For each activity $T_i \in T$, its minimum duration is denoted by d_i^{min} and its maximum duration by d_i^{max}. An activity can be non-interruptible (that is it has to be scheduled as one calendar event) or interruptible (can be split into multiple calendar events). The decision variable p_i denotes the number of parts in which the *i-th* activity has been split, with $p_i \geq 1$. In non-interruptible activities $p_i = 1$ holds.

T_{ij} denotes the *j-th* part of the *i-th* activity, where $1 \leq j \leq p_i$. A scheduled activity's duration should be between its minimum and maximum specified duration, that is the sum of the durations of all parts of an activity, denoted with d_i, must be at least d_i^{min} and no greater than d_i^{max}. For each T_{ij}, the decision variables t_{ij} and d_{ij} denote its start time and duration respectively. The sum of all d_{ij}, for every T_i, must equal d_i.

For each T_i, the minimum and maximum allowed part duration are denoted with $smin_i$ and $smax_i$ respectively. Similarly, the minimum and maximum allowed temporal distances between every pair of parts are denoted with $dmin_i$ and $dmax_i$.

For each activity T_i, its temporal domain D_i is defined as a set of temporal intervals $D_i = [a_{i1}, b_{i1}] \cup [a_{i2}, b_{i2}] \cup \ldots \cup [a_{i,F_i}, b_{i,F_i}]$, where F_i is the number of intervals of D_i. Obviously, all parts of T_i must be scheduled within its temporal domain.

A set of M locations, $Loc = \{L_1, L_2, \ldots, L_M\}$, as well as a two dimensional, not necessarily symmetric, matrix $Dist$ that holds the temporal distances between locations are given. Each activity T_i has a set of possible locations $Loc_i \subseteq Loc$, where its parts can be scheduled. The decision variable $l_{ij} \in Loc_i$ denotes the location where T_{ij} is scheduled.

Activities may overlap in time. Each activity T_i is characterized by a utilization value, $utilization_i$, such as $0 \leq utilization_i \leq 1$. Any scheduled activities sharing a time point must have compatible locations, that is locations with no temporal distance to each other. Moreover, the sum of their utilization values should not exceed the unit (the time point *maximum* utilization value).

The domain model supports four types of constraints over pairs of activities (the set C): Ordering constraints, minimum and maximum proximity constraints and implication constraints. An ordering constraint between two activities T_i and T_j, denoted with $T_i < T_j$, implies that no part of T_j can start its execution before all parts of T_i have finished. A minimum (maximum) distance constraint between activities T_i and T_j implies that every two parts, one of T_i and another of T_j, must have a given minimum (maximum) temporal distance. Finally, an implication constraint of the form $T_i \Rightarrow T_j$ implies that in order to include T_i in the plan, T_j should be included as well.

The above definitions present the constraints of the problem. Scheduling personal activities is considered a Constraint Optimization Problem. That said, the empty schedule is a valid schedule but with low utility (an empty schedule may have some utility due to implication preferences), thus we are interested in better schedules. The problem model provides 9 sources of utility for a plan π_i. The main source concerns the activities themselves. Each activity T_i included in the schedule contributes utility $U_i(d_i)$ that depends on its allocated duration. The way T_i is scheduled by a schedule π_i within its temporal domain constitutes another source of utility, $U_i^{time}(\pi_i)$. The user can define arbitrary functions of time over the temporal domain of each activity, thus it is possible to give higher utility when an activity's parts are scheduled in some morning, afternoon, evening, beginning of the week etc.

Some hard constraints have their soft counterparts, thus constituting additional utility sources. Particularly, minimum and maximum distance preferences between the parts of an interruptible activity contribute $U_{dmin_i}(\pi_i)$ and $U_{dmax_i}(\pi_i)$ respectively. Similarly, preferences can be defined over the way pairs of activities are scheduled (the set of these special preferences is denoted with P). These preferences are defined in the same way as their constraint counterparts, C, but with the addition of a utility value, for each preference, that is given to a plan that fully satisfies them. Particularly for ordering and proximity preferences, partial satisfaction of the preference is allowed. The Degree of Satisfaction for a partial preference p ($p \in P$), denoted with $DoS(p)$, is defined as the ratio of the number of pairs of infinitesimal parts (that is, parts of the smallest possible duration), one from T_i and another from T_j, for which the preference holds, to the total number of pairs of infinitesimal parts.

To summarize, the optimization problem 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 duration range, duration utility profile, temporal domain, temporal domain preference function, utilization, a set of alternative locations, interruptibility property, minimum and maximum part sizes as well as required minimum and maximum part distances for interruptible activities, preferred minimum and maximum part distances and the corresponding utilities.
(2) A two-dimensional matrix with temporal distances between all locations.
(3) A set C of constraints over pairs of activities (particularly ordering, proximity and implication constraints).
(4) A set P of preferences over pairs of activities (particularly ordering, proximity and implication preferences).

Schedule

the activities in time and space, by deciding the values of their start times t_{ij}, their durations d_{ij} and their locations l_{ij}, while trying to maximize the following

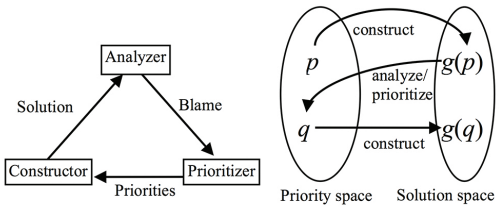


Fig. 1. (a) The SWO cycle. (b) Coupled search spaces

objective function:

$$U = \sum_{i}^{d_i \geq d_i^{min}} (U_i(d_i) + U_i^{time}(\pi_i) + U_i^{dmin}(\pi_i) + U_i^{dmax}(\pi_i)) + \sum_{p(T_i,T_j) \in P} u_p \times DoS(p(T_i,T_j)) \quad (1)$$

subject to the constraints of the problem model.[10]

2.2. *The SWO Approach*

In previous work,[10] the Constraint Optimization Problem is solved using the Squeaky Wheel Optimization (SWO) framework.[19] At its core, SWO uses a Construct/Analyze/Prioritize cycle as shown in Figure 1(a). The solution is found by a greedy approach, where decisions are based on an order of the activities determined by a priority queue. The solution is then analyzed to obtain the activities that cannot be scheduled. These activities' priorities are increased, thus enabling the constructor to deal with them earlier in the next iteration. This cycle will be repeated until a termination condition occurs.

SWO searches in two coupled spaces, as shown in Figure 1(b), these being the priority and solution spaces. Changes in the priority space cause changes in the solution space. Analyzing the previous solution and changing the order of the activities in the priority queue, causes changes in the priority space. A point in the solution space represents a possible solution to the problem.

One of the strengths of SWO is that it can easily be applied to new domains. The fact that small changes in the priority space can impact large ones on the solution space differentiates it from more traditional local search techniques such as WSAT.[20]

In subsequent work,[12] SWO has been extended with a post-optimization module that applies local-search techniques to further increase the final plan's quality. A bundle of transformations methods that utilize in-domain properties of the problem model have been employed. Coupled with an adaptation of the well known Simulated Annealing (SA) algorithm, that searches the space of locally reachable solutions

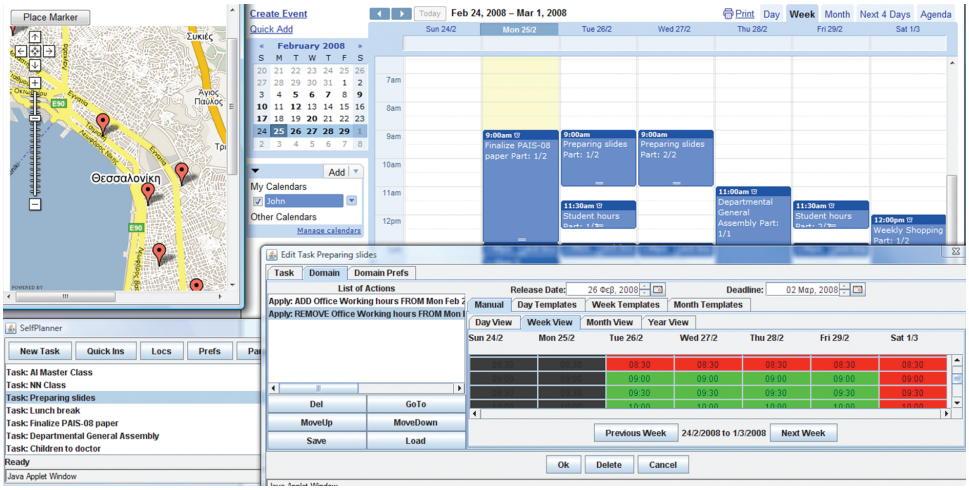


Fig. 2. User interface of SelfPlanner

(through the application of the transformations), the post-optimization module is able to improve the quality of SWO's output plans up to 20%.

2.3. SelfPlanner

This section gives an overview of the SelfPlanner system, including the new features introduced by this article. SelfPlanner[a] is a prototype system that has been designed as a web-based application. It is based on a client-server model, where the client can be accessed from a web-page and has been implemented in Java (as an applet), Javascript and PHP. It communicates with Google-based services (Google Maps and Google Distance Matrix), for location management and obtaining the traveling distance between locations.

The server portion of the application, where scheduling occurs, has been implemented in Java, and is responsible for communicating with the user, storing and managing his data, and formulating the user's problem for the scheduler, according to the model presented in the previous section. In addition, it communicates with the Google Calendar service, for importing a user's schedule into the system, and for exporting the calculated plan into his calendar. The scheduler itself (SWO plus SA post-optimization module) has been implemented in C++.

Figure 2 gives an overview of the SelfPlanner client application. The top-left browser window runs the locations-management subsystem where the user can define his locations. The bottom-left applet window is the main application, where the user can add new activities, edit existing ones, add a new constraints/preferences between pairs of activities etc, as well as request a plan for his problem instance.

[a]http://selfplanner.uom.gr

The bottom-right applet window is the Activity edit dialog, where the user can define or edit an activity, including its temporal domain. Having recognized from the early stages of SelfPlanner development that defining the temporal domain of an activity is a time-consuming task, the system offers significant flexibility in defining a temporal domain, comprising the design and application of various arbitrary templates (daily, weekly, monthly) combined with manual editing.[21] Finally, in the top-right browser window the user can view SelfPlanner's plan output on his Google Calendar.

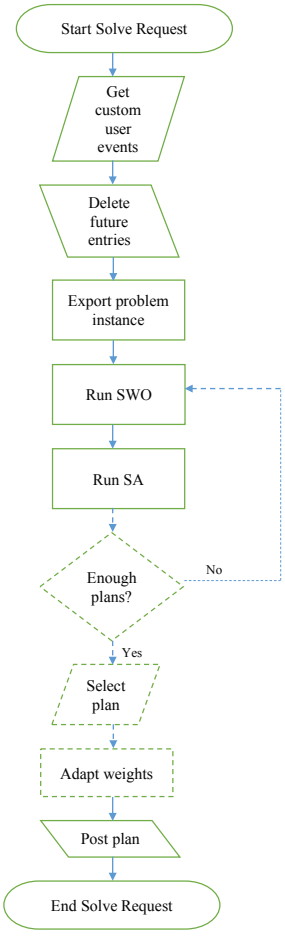


Fig. 3. SelfPlanner's plan generation process

Whereas SelfPlanner implements the problem model described in the previous section, it provides an extended higher-level view of it to the user. Some

extensions include the support of real dates[b] (versus discrete time points), easy definition of an activity's temporal domain, automatic location management, high-level temporal domain preferences, and periodic activities (the latter are implemented as one activity-per-period when calling the scheduler).

To use the system, the user should add her activities using the Activity Add/Edit dialog, which allows her to either add a new or edit an existing activity by defining the activity's name, duration, part durations, interruptible status, periodic status, temporal domain, temporal preferences, et. al. She can add constraints or preferences between pairs of activities using the Preferences dialog. She can define locations and attach them to activities using the locations-management subsystem. By pressing the "Solve" button she gets a user-specified number of alternative plans. If no one of the generated plans satisfies her, she can request more plans for the current problem instance or modify the current problem instance and solve it again. Note also that each time the user logs into the system or requests a new set of plans, she is asked whether all scheduled activity parts with elapsed start time (if any) have been executed or not; those that are indicated as executed do not participate in the subsequent problem solving requests. For a full description of the system see our previous work.[1]

Figure 3 presents a flow chart of SelfPlanner's plan generation process. Dashed shapes and lines indicate the new features firstly introduced by this article. Having specified her activities and constraints or preferences over them, the user can issue a solve request. SelfPlanner will first read the user-indicated Google calendars and get any custom events, i.e., non-SelfPlanner events that have been added directly in Google Calendar. These events are considered as busy time and their temporal interval is transparently removed from the domains of all SelfPlanner activities during the solving process. Next, the current problem instance is exported to the scheduler format, which is based on the problem model described in the "Problem Formulation" section, and the scheduler is called. The scheduler will produce a new plan, using the SWO algorithm and will subsequently post-optimize it using Simulated Annealing (SA).

The scheduling process will be repeated until the number of alternative plans requested by the user has been reached, while trying to maximize the deviation between those plans by using the methods presented in Section 3. Subsequently, the user will have to choose a plan among them. This will result in the weights used for learning the user's preferences being adapted by using the on-line learning method of Section 4. Finally, the system deletes all not-yet-started SelfPlanner generated entries from her Google Calendar, and updates her calendars with the selected plan (each activity is posted in a potentially different user-specified calendar).

[b]The resolution used is 30 minute intervals

3. Generating Alternative Plans

In this section we present our hybrid approach to generate qualitative, significantly different alternative plans. We define an enhanced evaluation function that takes into account the differences between the plan being evaluated and the already generated plans (if any).

3.1. *Hybrid Evaluation Function*

In order to generate an arbitrary number of qualitative, significantly different alternative plans, we extended the domain model by introducing the notion of the *value* of an alternative plan. This is defined as a linear combination of pure utility, as defined by formula (1), with the deviation of the plan under evaluation from the already generated plans. Let Δ be the set of the already generated plans, the value of a new plan π is defined by formula (2), as a replacement to formula (1):

$$V(\pi) = U(\pi) + c \times \frac{\sum_{\forall \pi' \in \Delta} U(\pi') \times PDiff(\pi, \pi')}{max(|\Delta|, 1)} \tag{2}$$

where c is a constant that weights the importance of π's difference over Δ's plans versus its utility, and $PDiff(\pi, \pi')$ is a function assessing the differences of any two plans π and π'. As it will be shown later in this section, $PDiff(\pi, \pi')$ ranges between 0 and 1, with higher values denoting greater differences between the two plans; $PDiff(\pi, \pi') = 0$ denotes no difference. By including factor $U(\pi')$ in the product in the numerator of Formula (2), new plans deviating from already generated high utility plans are favored over new plans deviating from already generated lower quality ones. Furthermore, the factor of $U(\pi')$ scales the second term of the sum to the scale of the first one. $|\Delta|$ in the denominator of formula (2) refers to the number of the already generated plans.

In case Δ is empty, that is no plan is already generated, formula (2) reduces to formula (1). Having found a nonempty set of plans, the scheduler tries to maximize concurrently the utility of the new plan, $U(\pi)$, as well as its average deviation from the already found plans, $U(\pi') \times PDiff(\pi, \pi')$, weighted by constant c.

Note finally that the hybrid evaluation function (2) is used only during the scheduling process. The generated plans are presented to the user ordered according to their pure utility, that is, $U(\pi)$.

3.2. *Running Example*

In order to demonstrate the new hybrid evaluation function and to clarify the deviation metrics that $PDiff(\pi, \pi')$ consists of, we introduce a running example consisting of the following activities:

- "Lecture" has to be scheduled on Wednesday evening, at 15:00, with a fixed duration of 2 hours, at the University.

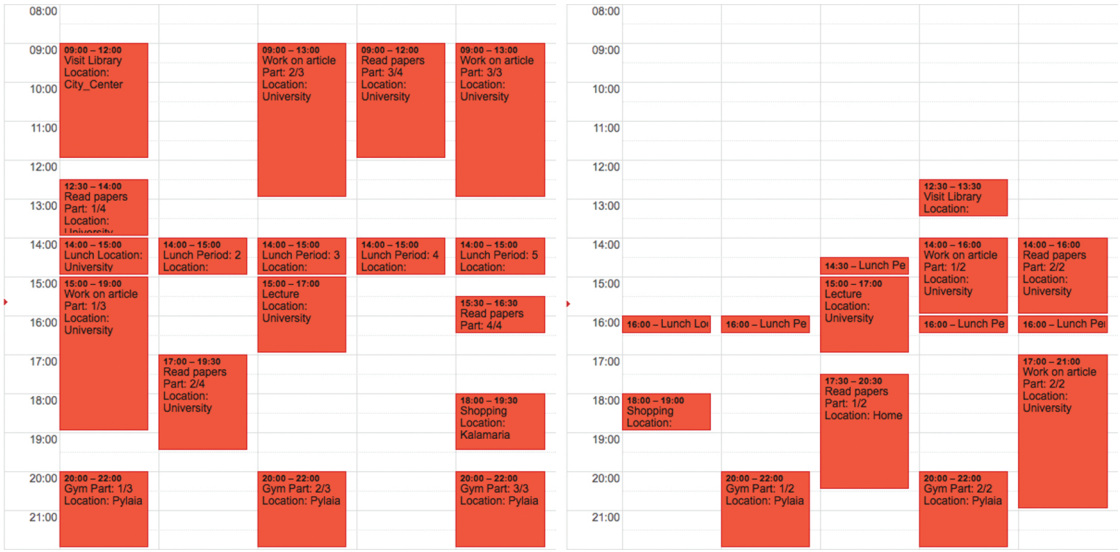


Fig. 4. (a) The main plan. (b) The alternative plan

- "Visit Library" has a variable duration between 1 and 3 hours, the library visiting hours (for the whole week) as its temporal domain, with the location of the library being downtown.
- "Shopping" has to be scheduled at the local supermarket, has a duration between 60 and 90 minutes, with the temporal domain being every day between 18:00 and 20:30.
- "Gym" is an interruptible activity located at the local gym, has a variable duration between 2 and 6 hours, which should be split in 2 hour parts being scheduled at least 24 hours away from each other. The temporal domain for this activity is every day between 19:00 and 22:00.
- "Work on Article" should be scheduled at the University, during the office working hours (09:00 to 21:00 daily); it lasts between 6 and 12 hours, which should be split in parts ranging between 2 and 4 hours each and be scheduled at least 24 hours away from each other.
- "Read papers" could be scheduled either at the University or at home, has a duration between 5 and 8 hours, which should be split in parts ranging between 1 and 3 hours, that should be 24 hours away from each other. The temporal domain for this activity is the office working hours.
- "Lunch" is a periodic non-interruptible activity that should be scheduled every day at the University, with a duration between 30 and 60 minutes and a temporal domain between 14:00 and 16:30.

We set a low value for the utility given to the extra duration of the activities. We defined two constraints: an implication constraint requiring the "Visit Library"

activity to be included in the plan if "Work on article" is also included; and an ordering constraint requiring the "Visit Library" activity to be scheduled before all parts of "Work on article."

We used SelfPlanner to produce two alternative plans for the running example, the first being the main plan that the system would normally produce and the second one being generated using hybrid evaluation function that takes into account its difference from the first one (Figure 4). The c constant of the hybrid evaluation function was set to 1.0. As we have given little utility for the extra duration, the most obvious difference of two plans is that the second one schedules the user activities closer to their minimum duration. The two plans are as different as they can be, with changes in activity durations, number of activity parts, different locations as well as scheduling times. In the next section, examples are given for the deviation metrics' role in the generation of the alternative plan.

3.3. *Quantifying the Degree of Deviation Between Two Plans*

We assess the deviation between two plans, π and π', with the use of function $PDiff(\pi, \pi')$, which takes into account the following aspects:

(1) The change in the total duration of each plan's activity.
(2) The change in the location of each plan's activity or part of it.
(3) The change in the time windows where the various parts of each activity have been scheduled.
(4) The change in the order in which pairs of activities have been scheduled.

In order to precisely define the above metrics, we introduce two extra functions. First we define the function $\tau(\pi, T_i, x) = t \in D_i$, which for a schedule π, an activity T_i and its x-th unit of duration, $1 \leq x \leq d_i$, maps it to the absolute time the x-th unit of duration is scheduled according to π. For example, assume that T_i has two parts, T_{i1} with $d_{i1} = 3$ and $t_{i1} = 2$, and T_{i2} with $d_{i2} = 2$ and $t_{i2} = 8$. Then, $\tau(\pi, T_i, 1) = 2$, $\tau(\pi, T_i, 2) = 3$, $\tau(\pi, T_i, 3) = 4$, $\tau(\pi, T_i, 4) = 8$ and $\tau(\pi, T_i, 5) = 9$. Similarly, we define the function $\lambda(\pi, T_i, x) = l \in Loc_i$, which maps the triple $\langle \pi$, T_i, $x \rangle$ to the location where the *x-th* unit of duration of T_i has been scheduled, according to π.

3.3.1. *Durations Deviation*

The total duration deviation between two plans π and π' is computed according to the following formula, which considers activities $T_i \in T$ that appear in at least one of the two plans:

$$\Delta D_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \text{ or } d'_i > 0}} \frac{|d_i - d'_i|}{max(d_i, d'_i)} \tag{3}$$

where d_i is the duration of activity T_i in plan π and d'_i is its duration in plan π'. ΔD ranges between 0 and 1.

Intuitively, the duration deviation measures the degree to which the activities have different durations in the two plans under comparison. In case all activities have the same duration in π and π', $\Delta D_{\pi,\pi'}$ is 0. In the other extreme, if any activity that is included in π is not included in π' and vice versa, then $\Delta D_{\pi,\pi'}$ is 1.

This metric's role is the easiest to spot on the alternative plan of Figure 4(b), since all the activity durations have changed. Since the system managed to schedule all the activities at their maximum duration in the main plan, whereas the utility of the extra duration is kept very low for all activities with variable duration, the alternative plan has all these activities with their minimum duration.

3.3.2. *Locations Deviation*

The total location deviation between two plans π and π' is computed according to the following formula, that considers activities $T_i \in T$ that appear in both plans:

$$\Delta L_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \text{ and } d'_i > 0}} \frac{1}{min(d_i, d'_i)} \times \sum_{\substack{x=1 \\ \lambda_x^{T_i} \neq \lambda'^{T_i}_x}}^{min(d_i,d'_i)} 1 \tag{4}$$

where $\lambda_x^{T_i} = \lambda(\pi, T_i, x)$ and $\lambda'^{T_i}_x = \lambda(\pi', T_i, x)$. $\Delta L_{\pi,\pi'}$ ranges between 0 and 1.

Intuitively, $\Delta L_{\pi,\pi'}$ measures the percentage of unit-duration portions of T_i that have different locations, by ordering them in each plan from the first to the last and matching them according to their order. In case T_i has different duration in the two plans, we arbitrarily consider the $min(d_i, d'_i)$ first unit-duration portions from T_i in both plans. In case all activities have the same location in both plans, or they are not included in both plans, $\Delta L_{\pi,\pi'}$ is equal to 0. In the other extreme case, when all activities are included in both plans and have totally different locations, $\Delta L_{\pi,\pi'}$ is equal to 1.

This metric's role in the alternative plan in Figure 4(b) can be observed on the "Read papers" activity, which can be scheduled either at the University or at home. In the main plan it was scheduled entirely at the University, whereas in the alternative plan one part was scheduled at home and the other parts remained at the University; the reason for that was to avoid unnecessary transfers between them and the adjacently scheduled activities (e.g., "Lunch").

3.3.3. *Absolute Time Deviation*

The absolute time deviation between two plans π and π' is computed according to the following formula, which considers activities $T_i \in T$ that appear in both plans:

$$\Delta Time_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \text{ and } d'_i > 0}} \frac{1}{min(d_i, d'_i)} \times \sum_{x=1}^{min(d_i,d'_i)} \frac{|\tau_x^{T_i} - \tau'^{T_i}_x|}{width_{D_i}} \quad (5)$$

where $width_{D_i} = b_{i,F_i} - a_{i,1}$ is the width of the temporal domain of T_i, $\tau_x^{T_i} = \tau(\pi, T_i, x)$ and $\tau'^{T_i}_x = \tau(\pi', T_i, x)$. $\Delta Time_{\pi,\pi'}$ ranges between 0 and 1.

Intuitively, $\Delta Time_{\pi,\pi'}$ measures the degree to which the unit-duration portions of T_i have been scheduled in different absolute time slots, by ordering these unit-duration portions from the first to the last and comparing each one to its corresponding (according to the order) in the other plan, in terms of their absolute time. In case T_i has different duration in the two plans, we arbitrarily consider the $min(d_i, d'_i)$ first unit-duration portions of T_i in both plans. In case all activities have the same schedule in both plans, or they are not included in both plans, $\Delta Time_{\pi,\pi'}$ is equal to 0. In the other extreme case, where all activities are included in both plans and have totally different schedules, that is, in the one plan they have been scheduled close to the leftmost part of their temporal domain, whereas in the other plan they have been scheduled close to the rightmost part of their temporal domain, $\Delta Time_{\pi,\pi'}$ tends to 1.

This metric's role in the alternative plan is very clear (Figure 4): No activity was scheduled at the same time it was scheduled in the main plan (except for the "Lecture" activity which had a fixed time).

3.3.4. *Ordering Differences*

For each pair of activities, T_i and $T_j \in T$, which both appear in a plan π, the precedence of T_i over T_j in π is computed as:

$$\nu_{ij} = \frac{1}{d_i \times d_j} \sum_{x=1}^{d_i} \sum_{y=1}^{d_j} \begin{cases} 1 \text{ if } \tau_x^{T_i} \leq \tau_y^{T_j} \\ 0 \quad \text{otherwise} \end{cases} \quad (6)$$

Intuitively, ν_{ij} represents the percentage of pairs of unit-duration portions, one from T_i and one from T_j, such that the portion of T_i is scheduled no later than the portion of T_j. ν_{ij} ranges between 0 and 1. Indeed, if T_i is totally ordered before T_j, then $\nu_{ij} = 1$, whereas if T_j is totally ordered before T_i, then $\nu_{ij} = 0$.

Furthermore, we extend the definition of ν_{ij} for the cases where both activities are not included in the plan. This is important because otherwise ν_{ij} might be defined in one plan and not in the other. So, if T_i appears in plan π but T_j does not appear in it, we define $\nu_{ij} = 1$. If T_i does not appear in π (irrelevant to whether T_j

appears in π or not), we define $\nu_{ij} = 0$. Subsequently, the ordering deviation for a pair of activities, T_i and T_j, in two plans π and π', is defined as:

$$\Delta O_{ij_{\pi,\pi'}} = |\nu_{ij} - \nu'_{ij}| \tag{7}$$

where ν'_{ij} refers to plan π'. $\Delta O_{ij_{\pi,\pi'}}$ ranges between 0 and 1.

Finally, we define the total ordering deviation as:

$$\Delta O_{\pi,\pi'} = \frac{2}{N \times (N-1)} \times \sum_{i=1}^{N} \sum_{j=i+1}^{N} |\nu_{ij} - \nu'_{ij}| \tag{8}$$

$\Delta O_{\pi,\pi'}$ also ranges between 0 and 1.

This metric's role in the alternative plan in Figure 4(b) can observed by comparing the ordering between the activities in the two plans. Most of the orderings have changed, with the exception of the case where an ordering constraint was defined.

3.4. *Plan Difference Between Two Plans*

Based on the above definitions, we define the difference between two plans π and π', $PDiff(\pi, \pi')$, as:

$$PDiff(\pi, \pi') = \begin{array}{c} \Delta D_{\pi,\pi'} \times W_D + \Delta L_{\pi,\pi'} \times W_L \\ +\Delta Time_{\pi,\pi'} \times W_{Time} + \Delta O_{\pi,\pi'} \times W_O \end{array} \tag{9}$$

where W_D, W_L, W_{Time} and W_O are non-negative weights, such as $W_D + W_L + W_{Time} + W_O = 1$. By specifying the values of these weights, we express our preferences on how we want new plans to deviate from already generated ones. For example, by setting all of them to 0.25, we express our preference about equal contribution of the four deviation metrics to the alternative plans. Alternatively, if we want the alternative plans to differ more in a specific aspect, we can set the corresponding weight at a higher value, reducing respectively the values of the other weights. Note finally that these weights do not correlate to the various sources of utility. Particularly, the user preference about how plans differ to each other is independent of his preference about how he evaluates a single plan.

4. Online Learning

In this section we present an online and unobtrusive machine learning method to address the problem of misstated user preferences. This method is strongly related to the multiple plan generation method presented in the previous section, as it relies on learning from the user's selections among the alternative plans presented to her. Particularly, each time the user selects a plan different from the optimal one produced by the multiple plan generation procedure, an error is assumed in her stated preferences and suitable adaptation factors are adjusted to better reflect the user's preferences.

In the adopted domain model,[10] utility of a plan arises from several sources, particularly:

(1) Activities included in the plan, with their minimum duration.
(2) Extra duration for the activities of item (1).
(3) The way an activity has been scheduled within its temporal domain.
(4) Minimum proximity preferences for the parts of interruptible activities.
(5) Maximum proximity preferences for the parts of interruptible activities.
(6) Ordering preferences.
(7) Minimum distance preferences.
(8) Maximum distance preferences.
(9) Implication preferences.

Defining preferences over all these aspects of a plan is a difficult and error-prone process for the user. So, it is very common for him to overweight some sources of utility and underweight some others. This form of systematic error, that has nothing to do with specific planning problems or activities, is what we attempt to learn and compensate in a non-intrusive manner, by monitoring the choices of the user among the alternative plans presented to him. Compensation is achieved through the introduction of nine compensation factors, named W_1 through W_9, each one of them corresponding to one of the nine aforementioned sources of utility. So, the total utility of a plan is the weighted sum of the utility aggregated from all sources, with the weights being the compensation factors. The initial values of the compensation factors are 1 (in that case, the sum of the various sources of utility coincides to Formula (1)), but they are changed by the online learning process, each time the user selects one of the alternative plans presented to him.

The online learning process works as follows: Having generated a set of alternative plans, they are presented to the user in decreasing order of their pure utility, taking the compensation factors into account. Let π be the first plan, that is, the best one according to the evaluation function and the current values of the compensation factors, and let π' be an alternative plan. If the user selects π', this is an indication that the evaluation function has an error, since the user has selected a plan evaluated as inferior to π. According to our assumption, this error has to do with the weighting of the various sources of utility, so we try to repair this error by changing the compensation factors.

So, let W_i, $1 \leq i \leq 9$, be the current values of the compensation factors, according to which π' is evaluated inferior to π, and let W_i' be their target values according to which π' is evaluated better than π (of course, there are multiple vectors of such 'correct' target values). Since learning is an online process, we want to change W_i's gradually towards to W_i', using a small learning rate.

The rationale behind learning the values of W_i is simple: If π' has higher utility than π with respect to utility source i, $1 \leq i \leq 9$, W_i is increased. If π' has lower utility than π with respect to some utility source i, W_i is decreased. Finally, if π' has the same utility as π with respect to utility source i, W_i is not changed. Furthermore, in order to change each compensation factor proportionately to the contribution of its utility source to the overall utility of the plans, we compute coefficient a_i as the

ratio of the contribution of utility source i in π' to the contribution of utility source i in π ($a_i > 1$ implies that W'_i should be increased, whereas $a_i < 1$ implies that W'_i should be decreased). So, the compensation factors are updated according to the following formula:

$$W'_i = \begin{cases} W_i, & \text{if } a_i = 1 \\ min(W_{max}, W_i \times (1 + Q \times (\alpha_i - 1))), & \text{if } a_i > 1 \\ max(W_{min}, W_i \times (1 - Q \times (1 - \alpha_i))), & \text{if } a_i < 1 \end{cases} \quad (10)$$

where Q is the learning rate, $W_{min} = 0.1$ and $W_{max} = 10$.

Example: Suppose there are only two utility sources, with π being better in utility source 1 and π' being better in utility source 2. Particularly, let $u_1 = 4$, $u'_1 = 2$, $u_2 = 2$ and $u'_2 = 3$ denoting the utility aggregated by each utility source for each one of the two plans respectively. Assume also that $W_1 = W_2 = 1$. With these values we have $U(\pi) = W_1 \times u_1 + W_2 \times u_2 = 6$ and $U(\pi') = W_1 \times u'_1 + W_2 \times u'_2 = 5$.

Suppose now that the user selects π' instead of π. We have $a_1 = 0.5$ and $a_2 = 1.5$. Assume also that $Q = 0.1$. Then, we have $W'_1 = W_1 \times (1 - 0.1 \times (1 - 0.5)) = 0.95$ and $W'_2 = W_2 \times (1 + 0.1 \times (1.5 - 1)) = 1.05$. With the new values of the compensation factors, $U(\pi) = 5.88$ and $U(\pi') = 5.05$. In other words, the difference in the evaluation between the two plans has been decreased in favor of π'. If the user consistently selects plans like π' over plans like π, after a number of iterations plans like π' will be evaluated as better than plans like π and will be presented earlier in the of the alternative generated plans.

However, we assume that the user is not static, that is, as time progresses he learns to define more accurately his preferences. So, each time the user selects the first plan, the compensation factors are attenuated, that is, they change towards their initial values. This happens according to the following formula:

$$W'_i = \begin{cases} W_i, & \text{if } W_i = 1 \\ W_i - \mu Q \times (W_i - 1), & \text{if } W_i > 1 \\ W_i + \mu Q \times (1 - W_i), & \text{if } W_i < 1 \end{cases} \quad (11)$$

where μ is the attenuation rate. This should be a much slower process.

The presented online learning process is based on the assumption that the error in evaluating the alternative plans is due to wrong compensation of the alternative high level utility sources. However, this is not always the case. For example, it might happen that π is better than π' in all sources of utility, so there is no way to make $U(\pi')$ higher than $U(\pi)$. Such strange cases arise because of errors at a more fine-grained level, that is, in compensating utility sources of the same type. As an example, consider the first source of utility, that is, the activities themselves. Suppose that the user assigns more utility to T_1 than to T_2, although he prefers mostly T_2 to T_1. This type of error cannot be learnt by the proposed approach. However, this type of error is not a systematic error, since in the next scheduling

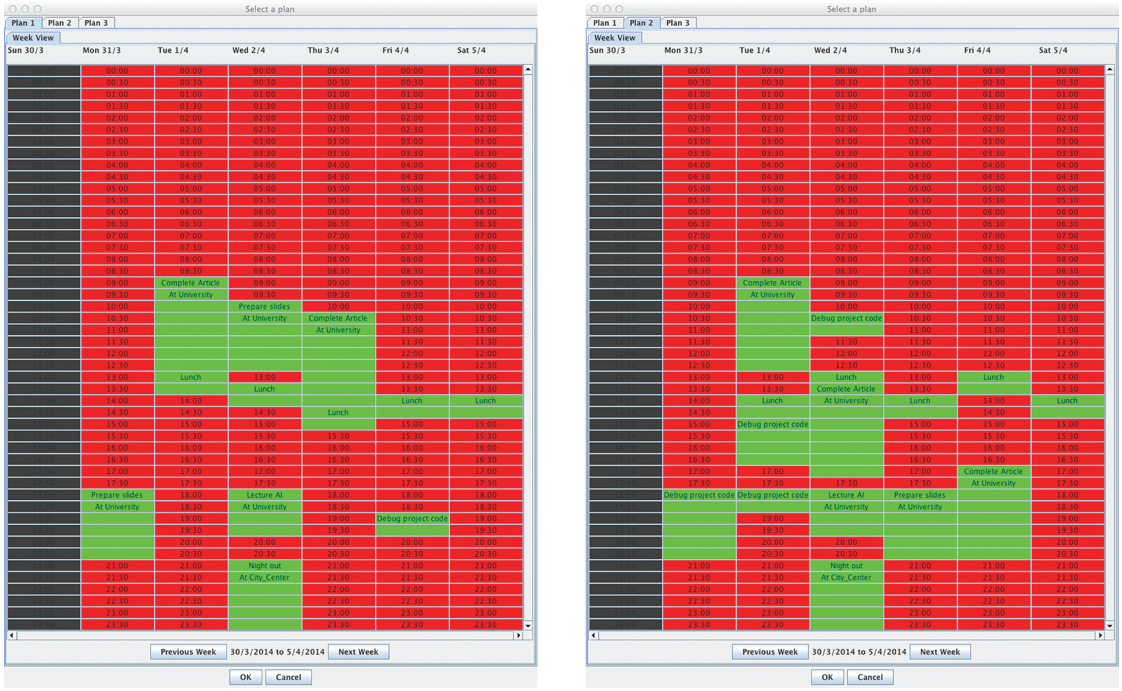


Fig. 5. Selecting a plan in SelfPlanner

problem, T_1 and T_2 will probably not be present. Such errors could be considered systematic by the introduction of ontologies. Indeed, if T_1 is an instance of class C_1, T_2 is an instance of class C_2 and the user systematically overestimates activities of class C_1 to activities of class C_2, this could be learnt by an online learning algorithm and compensated using suitable factors. This level of granularity constitutes our future work in online learning user preferences for personal activity scheduling.

5. Multiple Plan Support in SelfPlanner

We extended SelfPlanner to support the generation of alternative plans, as well as the online learning mechanism. Each time the user requests a new plan, a number of alternative plans are presented to him, using the current values of the compensation factors, as it has been described in the previous sections. The plans are sorted in descending order according to their weighted $U(\pi)$ values. When the user chooses his preferred plan, SelfPlanner adapts the compensation factors accordingly.

Figure 5 gives an overview of the new GUI features of SelfPlanner. Three alternative plans are presented in different tabs, with the main plan (Plan 1) being by default selected. For each plan, the user can see how his activities have been scheduled (green cells), as well as his free time (red cells). By pressing the "OK" button, his choice is recorded and the compensation factors are changed accordingly.

It is important to note that generating three alternative plans does not impose tripling the waiting time of the user. This is because a lot of the waiting time the user experiences when requesting a new plan is due to the communication with Google calendar, in order to read the user's other commitments, i.e., events directly entered into his Google calendar, and take them into account as busy time, when scheduling the SelfPlanner activities. This overhead remains the same, irrelevant to how many plans are produced by SelfPlanner. So, the only overhead (in terms of user satisfaction) for producing multiple plans, instead of one, concerns the increased scheduling time, which however in most cases is not noticeable at all.

For a full presentation of the system, the interested reader may refer to our previous work.[1]

6. Evaluation

This section evaluates the computational methods presented in the article. Particularly, it starts with evaluating the multiple plan generation process and continues with the online learning process. The source code for the complete scheduler and the experiments, as well as the problem definitions and test results are available online.[c]

6.1. *Generating Alternative Plans*

The SWO+SA scheduler (written in C++) was extended to support the modified evaluation function $V(\pi)$, which takes into account the distance from the already generated plans. After each successful execution of the scheduler, a dynamic list Δ, holding the already generated plans, is updated with the newly found plan. The list serves as an additional input parameter for the scheduler.

We tested the multiple plans generation algorithm on 35 random test cases, ranging in size from problem instances of 6 activities to 36, in steps of 5 activities, with 5 problems for each size. We set the parameters used for the alternative plan generation as follows: $c = 1.0$, a value that has been shown experimentally to produce balanced alternative plans in terms of quality and deviation; and $W_D = W_L = W_{Time} = W_O = 0.25$, thus denoting an equal preference on behalf of the user about how the new plans will deviate from the already generated ones, in terms of the four deviation metrics. As the scheduler uses a stochastic process for post-optimization,[12] we ran the above tests ten times for each problem and present the average values. From each run we obtained three plans. The learning process was disabled for these tests.

Figure 6 presents the average (over the 10 runs) utility change ratio for the two alternative plans over the first one, for each one of the 35 problem instances of the test set. The solid line represents the utility change percentage for the first

[c] *http://java.uom.gr/~talex/sch-mpe.zip*

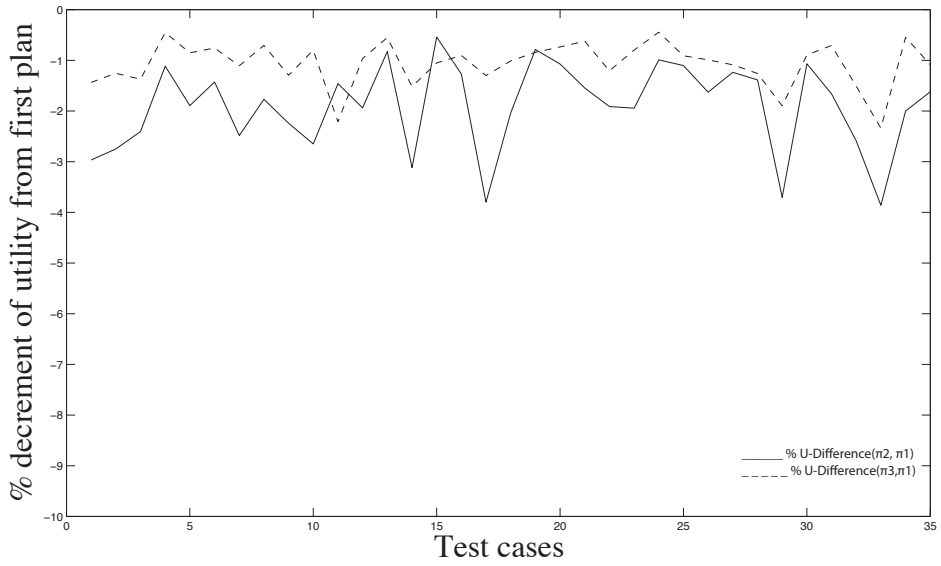


Fig. 6. Percentage difference of utility between the alternative plans and the original

alternative compared to the original, whereas the dashed line represents the utility change percentage for the second alternative, compared to the original. To compare utilities, the pure utility function $U(\pi)$ (Formula 1) is used; $V(\pi)$ is used for the generation phase only.

As it can be seen from Figure 6, the two alternative plans are of slightly lower utility than the original one. The utility change percentage for the first alternative plan, compared to the original, ranges from -3.86% to -0.54%, whereas for the second alternative the utility change percentage ranges from -2.33% to -0.44%. The average utility change percentage for the first alternative plan is -1.9% and for the second alternative is -1.06%. As it can be seen, the second alternative plan has on average higher utility than the first alternative. This is because the second alternative is generated while trying to differentiate both from the original and from the first alternative plans. This makes it difficult for the second alternative to differentiate significantly from the original, while maintaining a satisfying utility. Thus, usually, the second alternative lies somewhere between the original plan and the first alternative, both in terms of differentiation, as well as in terms of utility.

Figure 7 presents the *PDiff* values, that is the distance, for the two alternative plans compared to the already found plans in each case. Particularly, the first alternative plan is compared to the original plan, whereas the second alternative plan is compared to the original plan and the first alternative simultaneously. As it can be seen, *PDiff* values are significantly higher for π_2 compared to π_1, than for π_3 compared simultaneously to π_1 and π_2. The minimum *PDiff* value was 0.14 for

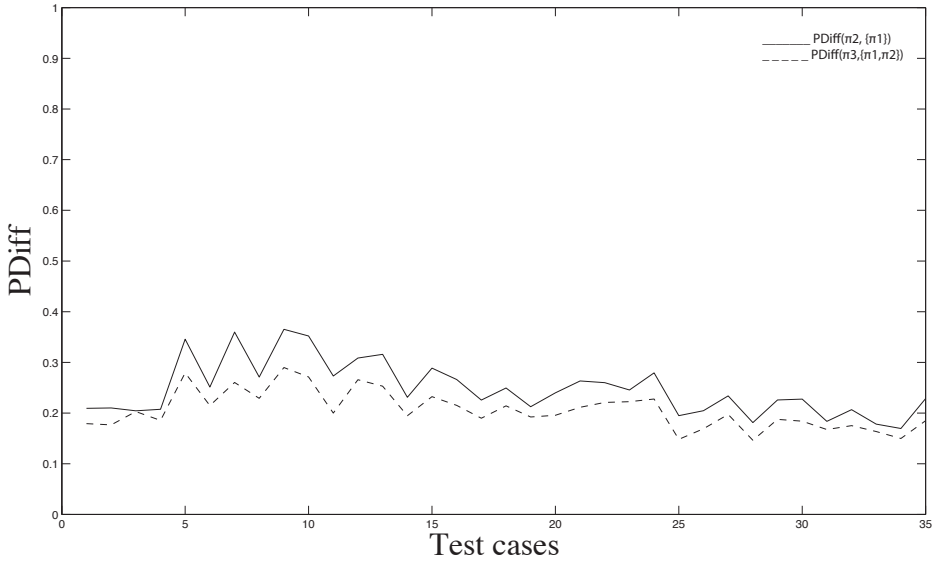


Fig. 7. PDiff values for the two alternative plans for each test case

the second alternative, whereas the maximum value was 0.36 for the first alternative. Another interesting observation is that *PDiff* values depend on the tightness of each problem, that is, the less activities within the same time period, the more chances for finding good alternative plans with significant differentiation. Indeed, in Figure 7 it is apparent a decrease in the *PDiff* values when moving to the right, i.e., towards problems with more activities.

A detailed view at the actual alternative solutions reveals apparent changes in many of the activities' parts temporal positions (the most common change), as well as their durations (another common change when applicable) and locations (rarest, as location selection do not provide a utility to standard SWO, except in the alternative plan generation phase). Changing the c parameter shifts the results either to more different plans of lower standard utility or to more similar plans to the original, thus with higher utility.

In order to demonstrate the relative effect of the values of the four deviation weights to the generated alternative plans, we ran an additional experiment with $W_{Time} = 0.7$ and $W_D = W_L = W_O = 0.1$. The results demonstrated on one hand noticeable less ordering, duration and location changes, but on the other hand far more rescheduling of activities at different times. $PDiff$ for the first alternative ranged from 0.28–0.52, being significantly increased compared to the the case where all deviation weights were equal. This could be addressed to the fact that, for the particular problem set, it is easier to change the start time of the activities than their ordering, location or duration

6.2. *Online Learning*

In order to evaluate the online learning process, we simulated two virtual users by quantifying their preferences randomly over each one of the nine sources of utility. Particularly, for each virtual user k, $1 \leq k \leq 2$, we selected a random value for each w_i^k in the range $[0.1, 10]$, where w_i^k is the k's virtual user actual compensation factor, with respect to the erroneous given model. These values are not known to the scheduler, so they represent the target of the online learning process.

Let $U'_k(\pi)$ be the modified version of $U_k(\pi)$, weighted by the k's virtual user preferences. These preferences are not known to the scheduler. So, the scheduler computes plans and attempts to guess the k's virtual user preferences using the online learning process presented in Section 4. For the k-th virtual user a random 30-activities problem is generated, the scheduler solves it and returns three plans to the user. These three plans are sorted in descending order according to the $U_k(\pi)$ values, using the estimated compensation factors, which are initially set to 1. Each time a set of three plans is returned to the k-th virtual user, the $U'_k(\pi)$ values of these plans are computed and the virtual user picks the one with the highest value, that is, according to its simulated preferences. If the plan picked by the k-th virtual user is not the same as the first plan returned by the scheduler, the compensation factors used by scheduler are adapted according to Formula 12. Then, a new random 30-activities problem is generated and the process is repeated until a termination condition occurs.

While testing the convergence of the online learning method to the hidden (for the scheduler) v_i^k values, the attenuation of the compensation factors, which occurs each time the user selects the first plan, was disabled. Attenuation serves no purpose when the user's preferences, as simulated by the v_i^k values, are static. So, when the virtual user picks the main plan returned by the scheduler, the compensation factors do not change.

The terminates condition fires when the user picks 30 times consecutively the first plan returned by the scheduler, in which case the simulation continues for twice so many iterations. Thus, for example, if the virtual user picked the first presented plan for $30th$ times consecutively at iteration 600, the simulation terminates at the $1200th$ iteration. This was chosen in order to better analyze the behavior of the learning process.

The simulation was performed twice for each virtual user, each time using a different value for the learning rate Q, particularly $Q = 0.1$ and $Q = 0.3$. For studying the converge of the learning process we defined the following metric:

$$m_i = \frac{\sum_{x=max(1,i-29)}^{i} \frac{U(\pi^x_{selected})}{U(\pi^x_1)}}{min(i, 30)} \tag{12}$$

where i is the problem instance being solved, $\pi^x_{selected}$ is the plan selected by the user for the $x-th$ problem instance, and π^x_1 is the first plan returned by the scheduler

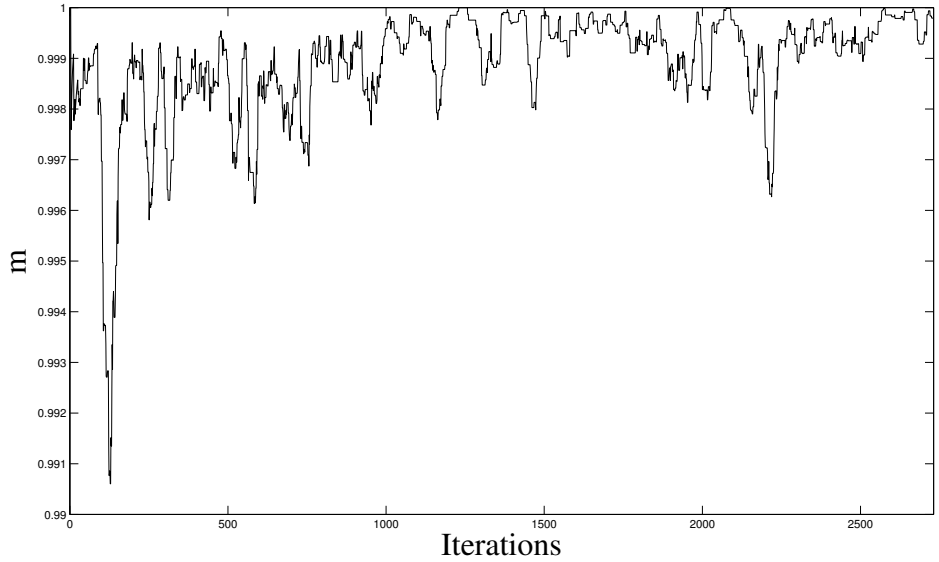


Fig. 8. Learning convergence for the simulation of Virtual User 1 with $Q = 0.1$

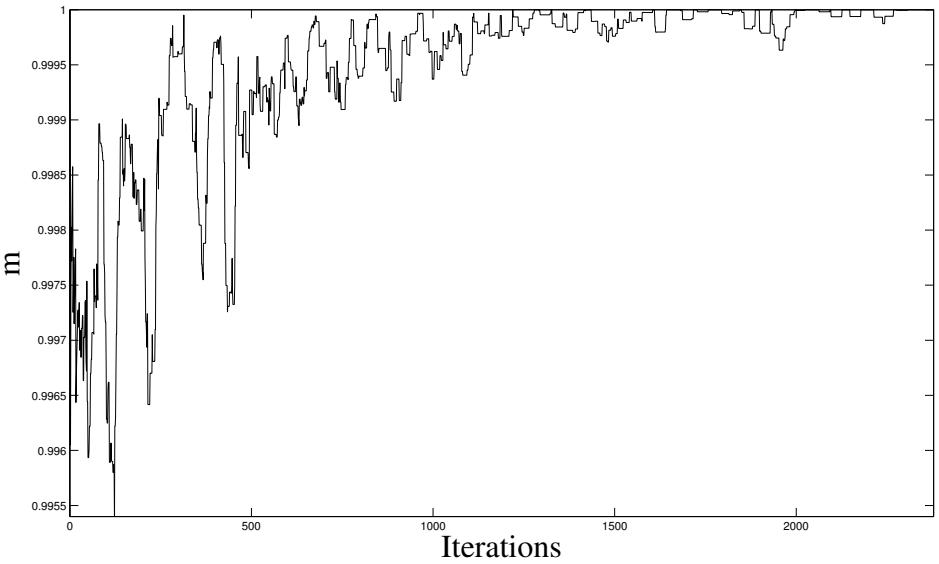


Fig. 9. Learning convergence for the simulation of Virtual User 1 with $Q = 0.3$

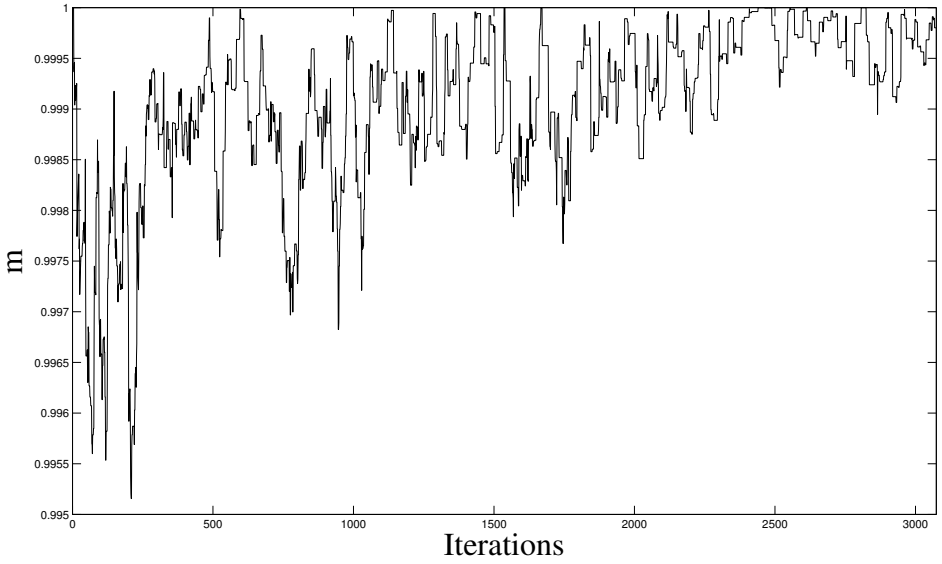


Fig. 10. Learning convergence for the simulation of Virtual User 2 with $Q = 0.1$

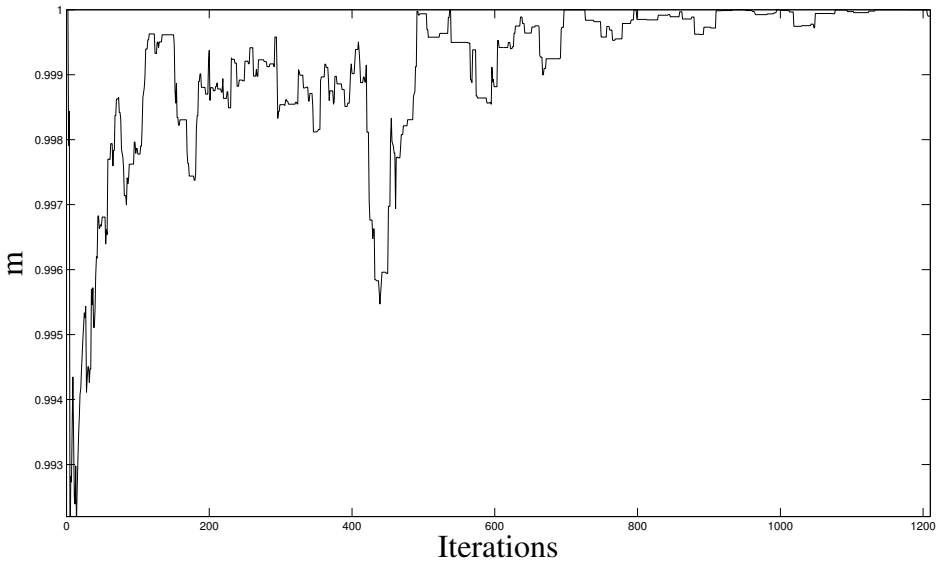


Fig. 11. Learning convergence for the simulation of Virtual User 2 with $Q = 0.3$

for the $x-th$ instance. This metric measures the relative quality of the (at most) last 30 plans selected by the user to the corresponding first plans suggested by the scheduler. If the user constantly selects the first plan suggested by the scheduler, then m_i tends to 1; on the contrary, if the user frequently selects plans of lower quality, m_i takes lower values. Thus, during online learning, we expect that m_i will start with lower values, which will increase towards 1 with the iterations.

Figure 8 presents the online learning method converging slowly to the first virtual user's preferences, with $Q = 0.1$, whereas Figure 9 demonstrates faster convergence for the same user and $Q = 0.3$. Similarly for the second virtual user, Figure 10 demonstrates slow convergence for $Q = 0.1$, whereas Figure 11 demonstrates faster convergence for $Q = 0.3$. All these figures demonstrate a behavior of the online learning method that is exactly what was expected, that is, the users select most frequently the first suggested plan, as online learning proceeds. Furthermore, the results demonstrate that better convergence, in terms of smoothness and stability, is achieved with $Q = 0.3$ than with $Q = 0.1$. The observation that the users do not constantly select the first suggested plan, has to do with the fact that the real values of the compensation factors have not been learned yet with infinite precision.

7. Conclusions and Future Work

In this article we presented new methods to generate an arbitrary number of qualitative, significantly different alternative plans for the problem of scheduling a user's individual activities, thus allowing the user to choose the one best suited to his needs. Our work is based on the assumption that a user will not always be able to specify his constraints and preferences correctly in the formal model—which has been found to be the case in many real situations. In such cases, offering a number of alternative plans to the user provides him with the possibility to pick the plan that best suits his actual preferences. The new methods are customizable through a number of parameters, enabling the user to choose which in-domain characteristics he considers more important, as well as to set his preference on the relative importance between plan variation versus plan quality.

In addition, we presented a non-intrusive online learning mechanism that attempts to reveal systematic errors in the way the user weighs his preferences over the various aspects of a plan. The mechanism is based on monitoring the user choices between the alternative plans and adapting the user preferences each time the user does not select the best, according to his expressed preferences, plan.

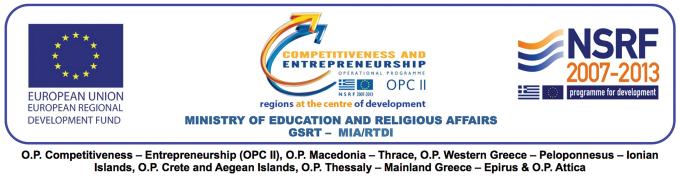
All these methods have been implemented in the prototype web-based intelligent calendar application SelfPlanner, a general purpose system aiming at helping a user to better organize his individual activities in time and space. Furthermore, the same technology supports the vertical application myVisitPlanner, which provides tourists with suggestions about cultural activities, as well as alternative plans for their daily tours inside a large geographical area.

Future work includes extending the problem formulation with some new at-

tributes, such as allowing the user to express his preferences over alternative locations, as well as employing activity ontologies so as to make it easier to express constraints and preferences over activities. The use of ontologies could also be exploited by the online learning mechanism, by allowing to reveal systematic errors performed by the user when evaluating alternative aspects of a plan, at the level of classes of activities. Concerning alternative plan generation, new metrics to measure the distance between two plans, either at the activity level or at the plan level, could be considered as well. Furthermore, we are actively working on combining all this work with meeting arrangements in a dynamic way, that is, using recursively the planning engine to reschedule already scheduled activities and meetings, in order to accommodate new meeting requests. Finally, we plan to conduct a large evaluation of the overall system, as well as of particular aspects of it, in terms of usability, intuitiveness, satisfaction, potential adoption, suggestions etc, employing real users and for a long period of time.

Acknowledgements

The authors are supported for this work by the European Union and the Greek Ministry of Education and Religions, under the program "Competitiveness and Enterprising" for the areas of Macedonia-Thrace, action "Cooperation 2009", project "A Personalized System to Plan Cultural Paths (myVisitPlanner^{GR})", code: 09SYN-62-1129.



References

1. I. Refanidis and A. Alexiadis. Deployment and evaluation of SelfPlanner, an automated individual task management system. *Computational Intelligence*, 27(1):41–59, 2011.
2. S. Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models In *Proceedings of the 21nd AAAI Conference on Artificial Intelligence*, pages 1601–1605, Vancouver, British Columbia, Canada, 2007.
3. P. M. Berry, M. Gervasio, B. Peintner, and N. Yorke-Smith. PTIME: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.*, 2(4):40:1–40:22, 2011.
4. K. Myers, P. Berry, J. Blythe, K. Conley, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe. An intelligent personal assistant for task and time management, *AI Magazine* 28(2):47-61, 2007.
5. J. S. Weber and M. E. Pollack. Entropy-driven online active learning for interactive calendar management. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pages 141–150, New York, NY, USA, 2007. ACM.

6. P. Viappiani, P. Pu, and B. Faltings. Preference-based search with adaptive recommendations. *AI Communications*, 21(2-3):155–175, 2008.
7. M. Freed, J. Carbonell, G. Gordon, J. Hayes, B. Myers, D. Siewiorek, S. Smith, A. Steinfeld, and A. Tomasic. RADAR: A personal assistant that learns to reduce email overload. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1287–1293, Chicago, USA, 2008. AAAI Press.
8. I. Refanidis. Managing personal tasks with time constraints and preferences. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, 2007*, pages 272–279. AAAI, 2007.
9. J. Bank, Z. Cain, Y. Shoham, C. Suen, and D. Ariely. Turning personal calendars into scheduling assistants. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, CHI EA '12, pages 2667–2672, New York, NY, USA, 2012. ACM.
10. I. Refanidis and N. Yorke-Smith. A constraint-based approach to scheduling an individual's activities. *ACM Transactions on Intelligent Systems and Technology*, 1(2):12, 2010.
11. A. Alexiadis and I. Refanidis. Meeting the objectives of personal activity scheduling through post-optimization. First International Workshop on Search Strategies and Non-standard Objectives (SSNOWorkshop 2012). CPAIOR, Nantes, France, 2012.
12. A. Alexiadis and I. Refanidis. Post-optimizing individual activity plans through local search. In *Proceedings of ICAPS 2013 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*, pages 7–15, Rome, Italy, 2013.
13. M. Roberts, A. Howe, I. Ray, and M. Urbanska. Using planning for a personalized security agent. AAAI Workshops, North America, 2012.
14. R. Dechter, N. Flerova, and R. Marinescu. Search algorithms for m best solutions for graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada, July 22-26, 2012*. AAAI Press, 2012.
15. A. Coman and H. Muñoz-Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, 2011*. AAAI Press, 2011.
16. T. A. Nguyen, M. B. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artif. Intell.*, 190:1–31, 2012.
17. K. L. Myers and T. J. Lee. Generating qualitatively different plans through metatheoretic biases. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 570–576, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.
18. A. Alexiadis and I. Refanidis. Generating alternative plans for scheduling personal activities. In *Proceedings of ICAPS 2013 Workshop on Scheduling and Planning Applications (SPARK)*, pages 35–40, Rome, Italy, 2013.
19. D. Joslin and D. P. Clements. Squeaky Wheel optimization. *J. Artif. Intell. Res. (JAIR)*, 10:353–373, 1999.
20. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, 1995.
21. A. Alexiadis and I. Refanidis. Defining a task's temporal domain for intelligent calen-

dar applications. In *Proceedings of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI)*, pages 399–406, Thessaloniki, Greece, 2009.