Optimizing Individual Activity Personal Plans through Local Search

Anastasios Alexiadis [a], Ioannis Refanidis [b]
[a] *Department of Applied Informatics, University of Macedonia,*
Egnatia str. 156, 54006, Thessaloniki, Greece
E-mail: talex@java.uom.gr
[b] *Department of Applied Informatics, University of Macedonia,*
Egnatia str. 156, 54006, Thessaloniki, Greece
E-mail: yrefanid@uom.gr

Optimization through local search is known to be a powerful approach to confront complex optimization problems. In this article we tackle the problem of optimizing individual activity personal plans, that is, plans involving activities one person has to accomplish independently of others, taking into account complex constraints and preferences. Recently, this problem has been addressed adequately using an adaptation of the Squeaky Wheel Optimization Framework (SWO). In this article we demonstrate that further improvement can be achieved in the quality of the resulting plans, by coupling SWO with a post-optimization phase based on local search techniques. Particularly, we present a bundle of transformation methods to explore the neighborhood of the solution produced by SWO using either hill climbing or simulated annealing. Similar results can be obtained by employing local search only, starting from an empty plan, thus demonstrating the strength of the proposed local search techniques. We present several experiments that demonstrate an improvement on the utility of the produced plans, with respect to the solutions produced by SWO only, of more than 6% on average, which in particular cases exceeds 20%. Of course, this improvement comes at the cost of extra time.

Keywords: Local search, personal activities, scheduling

Introduction

Electronic calendar applications and personal digital assistants typically involve sets of fully specified and independent events. These events constitute a user's schedule for a period of time. They are usually characterized by a fixed start time, a duration and, occasionally, a location where that particular event will take place. Furthermore, many systems support tasks. These are commitments potentially having a deadline to be met (e.g., writing an article or doing weekly shopping). Tasks are often kept separately, in task lists, and are not characterized by a specific start time. In some systems, as soon as a task is dropped into the calendar, it is automatically transformed into an event. The need to develop intelligent automated systems for calendar management has been identified by many researchers as an ambitious task for Artificial Intelligence [3,4,7,12,14,15,16].

In [18], a constraint optimization model for the problem of managing personal time is presented, treating events and tasks, referred as activities, in a uniform way. Each activity is characterized by a temporal domain, a duration profile, a set of possible alternative locations, preemptiveness (that is, possibility of being interrupted and resumed), utilization (that is, possibility of being performed concurrently with other activities), various preferences over its temporal domain and its duration profile, as well as constraints and preferences over the way parts of an interruptible activity are scheduled in time. The model also supports a variety of binary constraints and preferences, particularly ordering, proximity and implication relations. In the same work, a scheduler, based on the Squeaky Wheel Optimization (SWO) framework and coupled with domain-dependent heuristics, is employed to automatically schedule a user's individual activities. SWO is a powerful but incomplete search algorithm, so the solutions it produces are generally not optimal. This sub-optimality is more apparent in over-constrained, though practical problems, where SWO often produces schedules that a user can manually improve through simple transformations, such as scheduling an activity at a more preferred time point.

This last observation motivated us to improve upon SWO's solution quality. Post-optimization modules developed for planners recently, that exploit a plan's structure, have proven to be effective [5,6,11,13,19]. So, in this article we present local-search techniques that increase the quality of SWO's plan output through

AI Communications
ISSN 0921-7126, IOS Press.

post-optimization. In the particular setting, we devised and implemented a set of transformations of valid plans, such as shifting activities, changing their durations and locations, as well as merging or splitting parts of interruptible activities. Extensive experimental results have shown improvement over SWO's output up to 20.38%. In addition, we explore the contribution of each particular transformation on improving a schedule. Finally, we demonstrate that similar results can be achieved by commencing local search from the empty plan, thus demonstrating the strength of the designed bundle of transformations. However, post-processing dominates pure local search in hard problems, when the available computation time is bounded.

Earlier stages of this work have been presented in [1] and [2]. This article elaborates over previous work of the same authors in that (a) it advances the underlying stochastic local search algorithms by employing some form of stochastic backtracking during local search, as well as by relaxing the seed solution in overconstrained problems prior to local search; (b) it provides detailed experimental results, including evaluating the contribution of each individual transformation to the final output; (c) it illustrates the effectiveness of the proposed local search optimization algorithms through a realistic scenario; and (d) it discusses usability and transferability issues.

The rest of the article is structured as follows. First, we formulate the Constraint Optimization Problem and illustrate the SWO-based approach. We also briefly discuss usability issues. Next, we present the local search transformations to explore the neighborhood during the post-optimization phase, using either *hill-climbing* or *simulating annealing* [8,10]. Subsequently, we present experimental results over a large set of problem instances, taken from previous work [18]. Furthermore, we demonstrate the strength of the local search techniques when applied to an empty plan. Then we evaluate the contribution of each particular transformation on the post-optimization process. We conclude our evaluation by applying the proposed methods to a realistic scenario. Finally, we discuss the results, before summarizing the article and identifying directions of future work.

Background

In this section we outline the problem formulation, as well as the SWO approach adopted in [18] to cope with the problem. Furthermore, we briefly discuss usability issues.

Problem Formulation

Time is considered a non-negative integer, with zero denoting the current time. A set T of N activities, $T = \{T_1, T_2, \ldots, T_N\}$, is assumed. For each activity $T_i \in T$, its minimum duration is denoted with d_i^{min} and its maximum duration with d_i^{max}. No activity can be accomplished with a scheduled duration less than d_i^{min}. On the other hand, there is no point in allocating more than d_i^{max} to T_i. A greater duration in the range $[d_i^{min}, d_i^{max}]$ results in greater utility for the user.

The decision variable p_i denotes the number of parts in which T_i has been split, with $p_i \geq 1$ when T_i has been scheduled. T_{ij} denotes the *j-th* part of T_i, $1 \leq j \leq p_i$. Non-interruptible activities are scheduled as one part, that is, $p_i = 1$.[1] For each T_{ij}, the decision variables t_{ij} and d_{ij} denote its start time and duration. The sum of all d_{ij}, for a given i, is denoted with d_i, [2] which must be at least d_i^{min}, or 0 if T_i is not scheduled.[3]

For each T_i we define the minimum and maximum possible part duration, $smin_i$ and $smax_i$,[4] as well as the minimum and maximum allowed temporal distances between every pair of parts, $dmin_i$[5] and $dmax_i$.[6]

For each T_i, its temporal domain 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 these intervals . All parts of T_i must be scheduled within D_i. [7]

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. To each activity T_i a set of possible locations $Loc_i \subseteq Loc$, where its parts can be scheduled, is assigned. The decision variable $l_{ij} \in Loc_i$[8] denotes the particular location where T_{ij} is scheduled. Every pair of distinct activity parts T_{ij} and T_{mn} scheduled in different locations must have a minimum temporal distance of $Dist(l_{ij}, l_{mn})$ or $Dist(l_{mn}, l_{ij})$, depending on which part is scheduled earlier in time. [9]

Activities may overlap in time. This is supported through the introduction of utilization, which measures the degree an activity absorbs the attention of the person performing it. Particularly, each T_i is characterized by a utilization value, $utilization_i$, taking values between 0 and 1. At any time point, the set of scheduled activities should have compatible locations (that is, locations with no temporal distance to each other) and the sum of their utilization values should not exceed the one unit.[10]

The model supports four types of binary constraints: Ordering constraints, minimum and maximum proximity constraints and implication constraints. An order-

ing 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.[11] A minimum (maximum) distance binary constraint between activities T_i and T_j implies every two parts, one of T_i and another of T_j, must have a given minimum (maximum) temporal distance.[12, 13] 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.[14]

Alternative schedules may differ in their total utility, which is computed as the sum of the value accumulated from a variety of utility sources. The main source of utility concerns the inclusion of each particular activity T_i in the schedule and is denoted with $U_i(d_i)$, taking its minimum value when $d_i = d_i^{min}$ and its maximum value when $d_i = d_i^{max}$. A linear definition for $U_i(d_i)$ is the following:

$$U_i(d_i) = \begin{cases} 0, & \text{if } d_i < d_i^{min} \\ u_i^{low} + \frac{(d_i - d_i^{min})}{(d_i^{max} - d_i^{min})}(u_i^{high} - u_i^{low}), & \text{if } d_i^{min} \le d_i \le d_i^{max} \\ u_i^{high}, & \text{if } d_i > d_i^{max} \end{cases} \quad (1)$$

where u_i^{low} and u_i^{high} are constants with $0 \le u_i^{low} \le u_i^{high}$.

The way T_i is scheduled by a schedule π_i within its temporal domain constitutes another source of utility, $U_i^{time}(\pi_i)$, which is defined as:

$$U_i^{time}(\pi_i) = \frac{\sum_{j=1}^{p_i} \sum_{t=t_{ij}}^{t_{ij}+d_{ij}-1} u_i^{time}(t)}{d_i} \quad (2)$$

where $u_i^{time}(t)$ denotes the utility accumulated by scheduling one unit of time of T_i at t. Particularly, let t be a time slot in T_i's k-th interval $[a_{ik}, b_{ik})$, that is, $a_{ik} \le t < b_{ik}$, $1 \le k \le F_i$. Furthermore, we assume that for the arbitrary k-th interval of D_i, we are given two bound values, u_{ik}^{low} and u_{ik}^{high}. Then, $u_i^{time}(t)$ can be defined as:

$$u_i^{time}(t) = \begin{cases} 0, \\ \text{if } t < a_{ik} \text{ or } t \ge b_{ik}, \text{ for every } k \\ u_{ik}^{low} + (t - a_{ik})\frac{u_{ik}^{high} - u_{ik}^{low}}{b_{ik} - a_{ik} - 1}, \\ \text{if } t \ge a_{ik} \text{ and } t < b_{ik} \text{ for some } k \end{cases} \quad (3)$$

Any form of hard constraint can also be considered a soft constraint that might contribute utility. Partial satisfaction of ordering and proximity soft constraints is allowed. For example, minimum and maximum distance constraints between the parts of an interruptible activity contribute $U_i^{dmin}(\pi_i)$ and $U_i^{dmax}(\pi_i)$ respectively:

$$U_i^{dmin}(\pi_i) = \frac{\sum_{t \in \pi_i} \sum_{t' \in \pi_j, abs(t-t') \ge pdmin_i} 1}{d_i \times d_i} u_i^{dmin} \quad (4)$$

$$U_i^{dmax}(\pi_i) = \frac{\sum_{t \in \pi_i} \sum_{t' \in \pi_j, abs(t-t') \le pdmax_i} 1}{d_i \times d_i} u_i^{dmax} \quad (5)$$

where $pdmin_i$ and $pdmax_i$ are the preferred minimum and maximum distances between two parts of an interruptible activity ($pdmin_i > dmin_i$ and $pdmax_i < dmax_i$). According to the above formulas, $U_i^{dmin}(\pi_i)$ and $U_i^{dmax}(\pi_i)$ depend on the number of pairs of infinitesimal pieces (that is, corresponding to one unit of duration) of T_i, for which the proximity preference holds, to the total number of such pairs.

Similarly, binary preferences can be defined as well over the way pairs of activities are scheduled. Especially for ordering and proximity preferences, partial satisfaction of the preference is allowed. The Degree of Satisfaction of preference p (p being either an ordering, or a proximity preference), denoted with $DoS(p)$,

---

[1] $\forall T_i : p_i = \begin{cases} 0, & \text{if } T_i \text{ is not scheduled} \\ 1, & \text{if } T_i \text{ is not interruptible} \\ \ge 1, & \text{if } T_i \text{ is interruptible} \end{cases}$ (C1)

[2] $\forall T_i : \sum_{j=1}^{p_i} d_{ij} = d_i$ (C2)

[3] $\forall T_i : d_i^{min} \le d_i$ OR $d_i = 0$ (C3)

[4] $\forall T_{ij} : smin_i \le d_{ij} \le smax_i$ OR $d_{ij} = 0$ (C4)

[5] $\forall T_{ij}, T_{ik}\ j \ne k, d_{ij} > 0, d_{ik} > 0 :$
$t_{ij} + d_{ij} + dmin_i \le t_{ik} \vee t_{ik} + d_{ik} + dmin_i \le t_{ij}$ (C5)

[6] $\forall T_{ij}, T_{ik}\ j \ne k, d_{ij} > 0, d_{ik} > 0 :$
$t_{ij} + dmax_i \ge t_{ik} + d_{ik} \wedge t_{ik} + dmax_i \ge t_{ij} + d_{ij}$ (C6)

[7] $\forall T_{ij}, d_{ij} > 0 : \exists k, 1 \le k \le F_i, a_{ik} \le t_{ij} \le b_{ik} - d_{ij}$ (C7)

[8] $\forall T_{ij} : l_{ij} \in Loc_i$ (C8)

[9] $\forall T_{ij}, T_{mn}, T_{ij} \ne T_{mn}, d_{ij} > 0, d_{mn} > 0 :$
$(Dist(l_{ij}, l_{mn}) > 0 \vee Dist(l_{mn}, l_{ij}) > 0)$
$\Rightarrow t_{ij} + d_{ij} + Dist(l_{ij}, l_{mn}) \le t_{mn} \vee$
$t_{mn} + d_{mn} + Dist(l_{mn}, l_{ij}) \le t_{ij}$ (C9)

[10] $\forall t : \sum_{T_{ij}} utilization_i \le 1$ (C10)
$t_{ij} \le t < t_{ij} + d_{ij}$

[11] $\forall T_i, T_j, d_i > 0, d_j > 0 : T_i < T_j \Rightarrow$
$\forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0 : t_{ik} + d_{ik} \le t_{jl}$ (C11)

[12] $\forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0, dmin_{ij} > 0 :$
$t_{ik} + d_{ik} + dmin_{ij} \le t_{jl} \vee t_{jl} + d_{jl} + dmin_{ij} \le t_{ik}$ (C12)

[13] $\forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0, dmax_{ij} < \infty :$
$t_{ik} + dmax_{ij} \ge t_{jl} + d_{jl} \wedge t_{jl} + dmax_{ij} \ge t_{ik} + d_{ik}$ (C13)

[14] $\forall T_i, T_j : T_i \Rightarrow T_j \Leftrightarrow (d_i > 0 \Rightarrow d_j > 0)$ (C14)

is defined as the ratio of the number of pairs of infinitesimal pieces, one from T_i and another from T_j, for which the binary preference holds, to the total number of pairs of infinitesimal pieces.

$$DoS(p) = \frac{\sum_{t_i \in \pi_i} \sum_{t_j \in \pi_j, p(t_i,t_j) holds} 1}{d_i \times d_j} \quad (6)$$

For ordering and proximity preferences over pairs of activities, where not both activities are included in the current plan, $DoS(p) = 0$ holds. Implication preferences cannot be satisfied partially, so for them we have either $DoS(p) = 0$ or $DoS(p) = 1$.

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 profile, temporal domain and preference function over it, utilization, a set of alternative locations, interruptibility property, minimum and maximum part sizes as well as the 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 binary constraints (ordering, proximity and implication) over the activities.
4. A set P of binary preferences (ordering, proximity and implication) over the activities.

Schedule
the activities in time and space, by deciding the values of their number of parts p_i, their start times t_{ij}, their durations d_{ij} and their locations l_{ij}, while trying to maximize the following objective function:

$$U = \begin{array}{c} \sum_{i}^{d_i \geq d_i^{min}} (U_i(d_i) + U_i^{time}(\pi_i) \\ + U_i^{dmin}(\pi_i) + U_i^{dmax}(\pi_i)) \\ + \\ \sum_{p(T_i,T_j) \in P} u_p \times DoS(p(T_i, T_j)) \end{array} \quad (7)$$

subject to constraints (C1) to (C14).

The above formula computes the total global utility of a valid plan, which directly corresponds to the *plan quality*. It is also possible to compute a *loose upper bound* of a problem instance, by summing the maximum utilities of all preferences involved in that instance. However, since different preferences will usually contradict to each other, achieving their maximum utilities simultaneously is usually impossible.

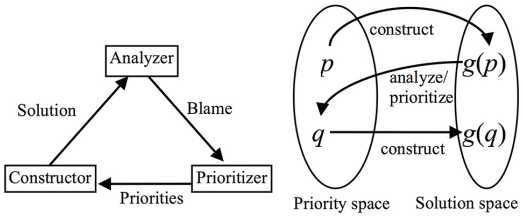


Fig. 1. (a) The SWO cycle. (b) Coupled search spaces

The SWO Approach

In previous work [18], the problem is solved using the Squeaky Wheel Optimization (SWO) framework [9]. 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. Their priorities are increased, enabling the constructor to deal with them earlier in the next iteration. The cycle is repeated until a termination condition occurs.

SWO is a fast but incomplete search procedure. The algorithm searches in two coupled spaces, as shown in Figure 1(b). These are the priority and solution spaces. Changes in the solution space are caused by changes in the priority space. Changes in the priority space occur as a result of analyzing the previous solution and using a different order of the activities in the priority queue. A point in the solution space represents a possible solution to the problem. Small changes in the priority space can impact large changes in the solution that is generated. However, not all solutions in the solution space are produced by some ordering in the priority space.

SWO can easily be applied to new domains. The fact that it gives variation on the solution space makes it different than more traditional local search techniques such as WSAT [20]. SWO was adapted to the Constraint Optimization Problem described in the previous section, using several domain dependent heuristics that measure the impact of the various ways that a specific activity is scheduled on the remaining ones. A solution is obtained by deciding values for the decision variables p_i, t_{ij}, d_{ij} and l_{ij}, for each T_i and T_{ij}, while trying to maximize Formula (7).

Usability issues

The complexity of the underlying model may arise usability concerns. There are two alternative approaches

postprocess-swo(*Activities*, *Best_Solution*)
  New_Best_Solution ← best_neighbor(*Activities*, *Best_Solution*)
  if U(*New_Best_Solution*) ≤ U(*Best_Solution*)
    return *Best_Solution*
  else
    return postprocess-swo(*Activities*, *New_Best_Solution*)
end

Fig. 2. Hill-climbing based post-processing algorithm

that can be adopted in order to address such concerns. The first one, which has been adopted by the SELF-PLANNER prototype application and has been evaluated in previous work [16], offers a rich user interface that abstracts many of the details of the underlying model, while hiding at a first glance some others. For example, high-level templates are used to define temporal domains, online maps are used to compute distances, etc.

The second approach is to provide ready-to-use activity descriptions, with all their details (e.g., temporal domain, location, duration, etc) pre-specified. This approach has been adopted by the MYVISITPLANER system [17], which focuses on cultural activities such as visiting museums and archaeological sites, attending performances, etc. For example, concerning a museum visit, the system knows when the museum is open, what is a typical duration for the visit, where is the museum, etc. Furthermore, MYVISITPLANER supports an ontology of activities, whereas the user can express his preferences over this ontology (as well as for particular activities). Thus, by selecting a cultural activity, MYVISITPLANER is able to schedule the activity within the user's calendar, while taking into account his preferences.

Of course, the two aforementioned approaches can be combined, as well as enhanced with machine learning techniques, in order to achieve even higher levels of usability.

Applying Local Search Methods to Improve SWO Output

In previous work [1] we applied a local search algorithm, based on *hill-climbing* (HC), to the output of SWO to further improve the solution quality. In more recent work [2] we extended this approach with the adoption of the more flexible *Simulated Annealing* (SA) stochastic search algorithm. This section focuses on the transformations that have been adopted by both post-optimization algorithms, assuming that HC is used. The following section presents the details of SA and how it elaborates over HC in the particular domain.

Figure 2 gives an overview of HC, as it has been applied to the problem of optimizing individual activity personal plans. U denotes the objective function of (7) and *Solution* and *New_Solution* denote complete and valid assignments to the decision variables p_i, t_{ij}, d_{ij} and l_{ij}. HC is initialized with the problem definition (*Activities*) and SWO's output solution as the initial *Best_Solution*.

Computing the Best Neighbor

The *best_neighbor* function computes all neighboring solutions by applying various transformations to the current solution and returns the best one of them. HC continues until no further improvement is possible the current solution; then, the last solution is returned.

Best Start Time

This transformation attempts to reschedule each part T_{ij} (with $d_{ij} > 0$) within D_i. Potential gains from rescheduling T_{ij} include:

1. T_{ij} is scheduled at a more preferred time window, that is, U_i^{time} increases.
2. T_{ij} is scheduled at a less congested time window, so d_i can increase (using another transformation at a later stage).
3. Other parts T_{kl} can be scheduled at a more preferred time window or they can increase their durations, using subsequent transformations.

The transformation attempts alternative values for each decision variable t_{ij} taken from its domain D_i, one value per decision variable at a time. For each value attempted, the constraints are checked to examine if the change is consistent with them. If it is not, the value is ignored.

Changing the Duration

This transformation attempts to change the duration of an activity part. Duration increment is a source of extra utility, but decreasing the duration of an activity part may provide extra temporal space to be exploited by subsequent transformations. HC does not accept duration decrements, since they result in a temporary utility drop; on the contrary, SA does accept them stochastically.

This transformation attempts different values for each decision variable d_{ij} (with current value $d_{ij} > 0$), ranging between $smin_i$ and $smax_i$. Each potential change is checked for constraint consistency.

Merging Two Parts of an Activity

This transformation attempts to merge two parts of the same activity into a single part. For part T_{ij}, with $d_{ij} > 0$, it iterates over all other parts of activity T_i, and for every T_{ik}, $j \neq k$ and $d_{ik} > 0$, it attempts to remove T_{ik} from the plan and to add its duration to T_{ij}. Combining two activity parts, T_{ij} and T_{ik}, into a single one (that is, reducing p_i by one), provided that duration constraints are not violated, may provide temporal space for other activity parts to be scheduled, which could not fit in the schedule up to the application of this transformation.

Particularly, for every ordered pair (T_{ij}, T_{ik}), $j \neq k$, $d_{ij} > 0$ and $d_{ik} > 0$, the transformation first checks whether $d_{ij} + d_{ik} \leq smax_i$ holds. In case it does, it adds d_{ik} to d_{ij}, sets $d_{ik} = 0$ and attempts to produce two new solutions: One in which t_{ij} does not change (that is, the extra duration is added after T_{ij}) and another in which d_{ik} is subtracted from t_{ij} (that is, the extra duration is added before T_{ij}). As usual, every solution produced is checked for constraint consistency.

Transfering Duration

This transformation attempts to transfer duration between two parts of the same activity. For every ordered pair of parts, (T_{ij}, T_{ik}), $j \neq k$, it attempts to transfer duration from T_{ik} to T_{ij}. Particularly, for every T_{ik}, such as $smin_i < d_{ik}$ and $d_{ij} < smax_i$, up to four neighboring solutions are computed:

1. Move the maximum possible duration, δd^i_{kj}, from the end of T_{ik} to the end of T_{ij}.
2. Move the maximum possible duration, δd^i_{kj}, from the end of T_{ik} to the beginning of T_{ij}. The start time of T_{ij} is updated by $t_{ij} = t_{ij} - \delta d^i_{kj}$.
3. Move the maximum possible duration, δd^i_{kj}, from the beginning of T_{ik} to the beginning of T_{ij}. The start times of both parts are updated by $t_{ij} = t_{ij} - \delta d^i_{kj}$ and $t_{ik} = t_{ik} + \delta d^i_{kj}$.
4. Move the maximum possible duration, δd^i_{kj}, from the beginning of T_{ik} to the end of T_{ij}. The start time of T_{ik} is updated by $t_{ik} = t_{ik} + \delta d^i_{kj}$.

The maximum possible duration to move from T_{ik} to T_{ij} is initialized with $\delta d^i_{kj} = min(smax_i - d_{ij}, d_{ik} - smin_i)$. In all cases, the durations of two parts are changed by $d_{ik} = d_{ik} - \delta d^i_{kj}$ and $d_{ij} = d_{ij} + \delta d^i_{kj}$. If the constraint consistency check on any of the four cases fails, δd^i_{kj} is recursively decreased by one and the transformation is reapplied for this particular case until either a valid solution is found or $\delta d^i_{kj} = 0$.

Splitting a Part

This transformation attempts to split a part into two parts. For part T_{ij}, where $d_{ij} \geq 2 \times smin_i$, it attempts to create a new part T_{ik} where $k = p_i + 1$. The new part will have $d_{ik} = smin_i$ and $l_{ik} = l_{ij}$, whereas d_{ij} is decreased by $smin_i$. Alternative values are tried for t_{ik}, trying to satisfy the problem constraints. If no valid t_{ik} is found, no new solution is produced; otherwise, one solution for each possible t_{ik} can be produced (HC would select the best one of them; SA would select any of them stochastically).

Increasing the Duration of an Activity

This transformation increases the duration of an activity part T_{ij} and, consequently, of the whole activity T_i, by one time unit, thus increasing $U_i(d_i)$. Particularly, for each part T_{ij}, where $d_{ij} < smax_i$ and $d_i < d_i^{max}$, it attempts to produce up to two new solutions. In both cases, d_{ij} is increased by one. In the first case, t_{ij} does not change, that is, the extra duration for T_{ij} is added after it. In the second case, t_{ij} is reduced by one, that is, the extra duration for T_{ij} is added before it.

Swapping Parts of Different Activities

This transformation attempts to swap the start times of two parts, T_{ij} and T_{kl}, $i \neq k$. Sometimes, an increase in the overall utility is possible in this way. This swap may not be possible with the previous transformations, because it would require moving temporarily either T_{ij} or T_{kl} in a less preferred time window (not allowed by HC), or because there is no free space to move temporarily one of the two parts; hence the necessity to have an explicit transformation for such swaps. As usually, each potential new solution is checked for constraint consistency.

Adding a Part

Having applied several transformations, it may be possible to add an extra part for an already scheduled activity T_i, thus significantly increasing $U_i(d_i)$. This transformation is different from the previous ones in that it does not operate on parts but on whole activities. For each activity T_i, with $d_i^{min} \leq d_i < d_i^{max}$, it is attempted to add a new part in the current schedule, provided that:

$$d_i \leq d_i^{max} - smin_i \tag{8}$$

that is, adding a new part with the minimum possible duration for the pair does not exceed the maximum allowed duration d_i^{max} for the whole activity.

For the new part T_{ik} ($k = p_i + 1$), for each possible t_{ik} where the new part could be scheduled, l_{ik} is greedily selected from Loc_i in such a way that traveling time from the locations of the adjacent (with respect to t_{ik}) parts is minimized, whereas d_{ik} is set to the maximum possible duration of part T_{ik} for the specific values of t_{ik} and l_{ik}. The best combination of (t_{ik}, l_{ik}) is returned. If adding a new part is successful, p_i is increased by one.

Adding an Activity

Similarly to *Adding a part*, it may be possible to add a whole new activity (that is, an activity that was not possible to be scheduled previously) to the schedule. For an activity T_i that is not included in the current schedule, this transformation greedily tries to schedule it by inserting parts to the timeline within its domain, starting from the earliest and proceeding to the latest time points. At each time window where a part of T_i can be inserted, it is inserted with the maximum possible duration and at the best possible location, taking into account the locations of the two already scheduled temporally adjacent activities. The transformation terminates as soon as:

$$\sum_{j=1}^{p_i} d_{ij} \geq d_i^{min} \tag{9}$$

is satisfied. Due to the greedy strategy of this transformation, it might fail to schedule an unscheduled activity, although it was possible to schedule it.

Changing Locations

Changing the location l_{ij} of part T_{ij} does not immediately affect the utility of a plan, since the model does not assign utilities to the alternative locations of an activity. However, it may increase the free time that is available for scheduling other activities, by reducing the time needed for traveling.

This transformation produces a new solution for every location change. Particularly, for every scheduled part T_{ij}, this transformation attempts to change its location, by trying every alternative (if any) location l in Loc_i. The best location selected is the one that minimizes the *total traveling time* of the plan. The total traveling time of a plan is calculated as the sum of all time intervals reserved for traveling between locations. In case of finding a better location for the current part, this transformation triggers all the aforementioned transformations that concern parts of scheduled activities.

Trimming Activity Parts

The aforementioned transformations work better if plenty of free time is available. There are cases though where the activities have tight domains and the post-processor has to operate upon a *tight* solution as its initial input. In such cases, the HC algorithm usually terminates with no solution improvement, whereas SA usually manages to improve it by first applying the *Change Duration* transformation and decreasing some activities' duration.

This transformation attempts to free some time on tight plans. It is called once only on a seed solution before it is passed to the post-processor for improvement. Initially, it determines the *tightness* of SWO's solution using a heuristic approach. Particularly, the utilization values of all used time slots (either for scheduled activity parts or for traveling time) is summed, and divided by the net size of the available domain for all activities. If the plan tightness is greater than a threshold (we used 0.9), trimming is applied on SWO's solution before it is passed to the post-processor. Trimming works as follows: For every T_{ij}, starting from $j = 1$, it sets its duration d_{ij} to the value $max(smin_i, d_{ij} - d_i + d_i^{min})$, that is, to the minimum possible duration such that the total duration of T_i becomes no less than d_i^{min}.

Avoiding Local Optima

The *hill-climbing* post-processing algorithm can get stuck in local optima. On the other hand, *simulated annealing* [10] alleviates this problem by giving a chance (although with low probability) to escape from local optima. So, to advance our previous work, we replaced hill-climbing as a post-processing algorithm with *simulated annealing* (SA), empowered with a tabu list and backtracking.

Similar to HC, SA based post-processing algorithm (Figure 3) uses as seed the solution produced by the preceding SWO phase. The *transformation functions* presented in the previous section have been modified so as to return all the valid neighbor solutions, instead of just the best one. A solution is picked stochastically in four stages: First a transformation is selected, then an activity (scheduled or not scheduled, depending on the transformation) is selected, then a part of an activity is selected (if anticipated by the transformation) and, finally, all the valid neighbors for the selected transformation, activity and part of it are computed and one of them is picked.

The SA post-processing algorithm uses the *random_neighbor* function (instead of *best_neighbor*), which

enhanced-postprocess-swo(Activities, Solution, Best, K, Tabu, SCHEDULE, KMAX, EMAX)

if K $\geq$ KMAX OR U(Best) $\geq$ EMAX
return Best
if $|Tabu|$ = $\frac{\text{KMAX}}{10}$
Tabu $\leftarrow$ Tabu $\setminus$ (Oldest_Solution $\in$ Tabu)
Tabu $\leftarrow$ Tabu $\cup$ {Solution}
T $\leftarrow$ SCHEDULE[K]
$C \leftarrow \emptyset$
DO i $\leftarrow$ 0 to ∞
if i+ K = KMAX
return Best
New $\leftarrow$ random_neighbor(Activities,Solution,C), New $\notin$ Tabu, $(C_{ikt} \in C) \leftarrow (C_{ikt} \in C) \setminus$ New
if New = NULL
select S out of $Tabu$ stochastically favoring older solutions
return enhanced-post-process-swo(Activities, S, Best, K, Tabu $\setminus S$, SCHEDULE, KMAX, EMAX)
$\Delta E \leftarrow U(New) - U(Solution)$
UNTIL $\Delta E > 0$ OR select New with probability $e^{\frac{\Delta E}{\text{T}}}$
K $\leftarrow$ K $+ i + 1$
if New $>$ Best
Best $\leftarrow$ New
return enhanced-post-process-swo(Activities, New, Best, K, Tabu, SCHEDULE, KMAX, EMAX)

end

Fig. 3. Simulated annealing based post-processing algorithm

returns a random neighbor state. For increased efficiency, the sets of neighbor solutions created are cached as long as the randomly picked neighbor solutions are not accepted (variable C_{ikt} in Figure 3 holds the cached neighbors for activity T_i, part T_{ik} and transformation t). Thus, if one of the cached triples (transformation, activity, part) is selected in the next iteration, the set of neighbor solutions for that transformation is not generated again. Cached data are cleared as soon as a new solution is accepted. The number of valid states, for a transformation applied on a part, is not known beforehand.

If the selected variable list C_{ikt} is empty, the algorithm will choose another transformation or activity part. If all the lists are empty, the search algorithm will backtrack to a previous solution, according to a probability distribution. Thus, *backtracking* is employed each time SA cannot find a neighbor that is not already in the tabu list. Without backtracking, such situations constitute a termination condition for SA. Backtracking selects stochastically a solution from the tabu list, giving higher probability to earlier solutions, and continues from there on. Backtracking is employed in over-constrained problems, when the size of the neighborhood decreases drastically towards the size of the tabu list.

The SA search algorithm uses four parameters: KMAX, EMAX, SCHEDULE and $|Tabu|$. KMAX is the number of iterations of the simulated annealing algorithm and EMAX is the loosely computed upper bound of the utility for the problem in hand. SCHEDULE defines the cooling schedule of SA. We used a cooling schedule dependent on KMAX, which is defined as: SCHEDULE[K] = SCHEDULE[K-1] $\times$ $(1 - 0.07 \times \frac{100}{\text{KMAX}})$, where SCHEDULE[1] = 0.9. We set the maximum value of $|Tabu|$ (the number of past states kept in the tabu list) to $\frac{\text{KMAX}}{10}$. These values were chosen after extensive experimentation.

Evaluation

This section comprises an extensive evaluation of the proposed local search techniques applied to the problem of scheduling individual personal activities. First we compare the base algorithm SWO to its enhancements with either HC or SA in the post-processing phase, employing the eleven transformations presented in the previous section. We also compare SWO to SA applied to an empty plan. Then, we present a sensitivity analysis showing the contribution

of each individual transformation to the performance of SA. Next we evaluate post-processing optimization on an illustrative realistic scenario. Finally, we discuss transferability issues of the evaluated computational methods.

Comparing to SWO

We first compare the original SWO algorithm to SWO coupled with HC at a post-processing phase, as well as to SWO coupled with SA, on 60 test cases, ranging in size, taken from previous work [18]. All experiments involving either HC or SA were executed 10 times and the averages are reported. The implementation of the above algorithms was done in C++ and the experiments were run on an Intel Xeon 2.66GHz processor.

According to the literature [18], the problems in the test suite differ only in the number of activities, whereas the rest of the parameters are constant. The planning horizon is set to 500 time units for all problems, with the temporal domain of each activity being a random set of intervals covering the entire planning horizon. 70% of the activities have fixed duration. 40% of the activities are interruptible with minimum distance constraints, 30% of these have maximum distance constraints, 30% have minimum distance preferences and 30% have maximum distance preference. 20% of the activities have utilization 0.5; the rest have full utilization. Binary constraints and preferences are defined with probability $1/(2N)$ for every pair of activities and every type of constraint or preference. There are 4 locations, with random subsets of them assigned to the activities. Activity utilities are selected from the interval $[5,12]$, temporal preference utilities from the interval $[5,10]$ and duration utilities from the interval $[2,5]$. All utilities assigned to binary preferences are selected from the interval $[1,3]$.

The results of the first comparison are shown in Figure 4. The plot represents the plan quality percentage improvement of the post-processing algorithms to the original SWO. We explored five post processing configurations: SWO followed by HC (denoted with SWO+HC); SWO followed by SA running for 2,000 iterations (denoted with SWO+SA2K); SWO+SA2K followed by HC, in order to reach the nearest local maximum (denoted with SWO+SA2K+HC); SWO followed by SA running for 50,000 iterations (denoted with SWO+SA50K); and, finally, SWO+SA50K followed by HC (denoted with SWO+SA50K+HC).

As we can observe from Figure 4, there is an improvement in all test cases with the less aggressive configurations SWO+HC and SWO+SA2K. The best case was a 18.46% improvement in plan quality with SWO+SA2K. The average improvement was 3.75% for SWO+HC and 5.37% for SWO+SA2K. SWO+SA2K+HC's improvement over SWO was 5.89% on average. Concerning the more aggressive configurations, SWO+SA50K+HC improves over SWO 7.51% on average, with a best case of 20.38%, whereas SWO+SA50K's average improvement was 7.48%, with a best case of 20.37%.

Figure 5 compares the run-time of the algorithms. The figure is in logarithmic (base 10) scale, with the yellow line representing SWO as the base case. Concerning the less aggressive configurations (SWO+SA2K and SWO+HC) on relatively small instances, that is, with up to 20 activities, the run-times of them are similar. On larger instances, SWO+SA2K outperforms SWO+HC in terms of run-time, whereas simultaneously it provides better quality results as it has been shown in Figure 4. Having in mind real-world situations, we consider the time requirements acceptable, since the problems of the test set are artificially created with increased complexity (many binary constraints and preferences), thus they are more complex than typical real-world situations that are expected to involve less activities with few interdependencies between them.

Concerning the more aggressive configurations,SWO +SA50K and SWO+SA50K+HC have similar runtimes, since there are not many chances for further improvement after running SA for 50K steps. Finally, SWO+SA2K+HC is on average 114% more time demanding than SWO+SA2K.

In addition, we compared the performance of the above algorithms to the *experimental upper bound* of the various problem instances. The *experimental upper bound* was computed by running the SWO+SA50K +HC algorithm 170 times on the test set and obtaining the utility of the best solution found for each problem instance. These results are shown in Figure 6. Particularly, SWO had an average plan quality of 92.58% to the experimental upper bound, whereas SWO+HC and SWO+SA2K had 95.99% and 97.47% respectively. Coupling SA with HC resulted in less than 1% of average improvement relative to SA (97.94%), whereas increasing KMAX raised it to 99.39%.

The worst case for SWO had a plan quality of 82.59%, which was increased to 95.89% with SWO +HC, and to 97.84% with SWO+SA2K. SWO+SA50K +HC further raised it to 99.43% on average.

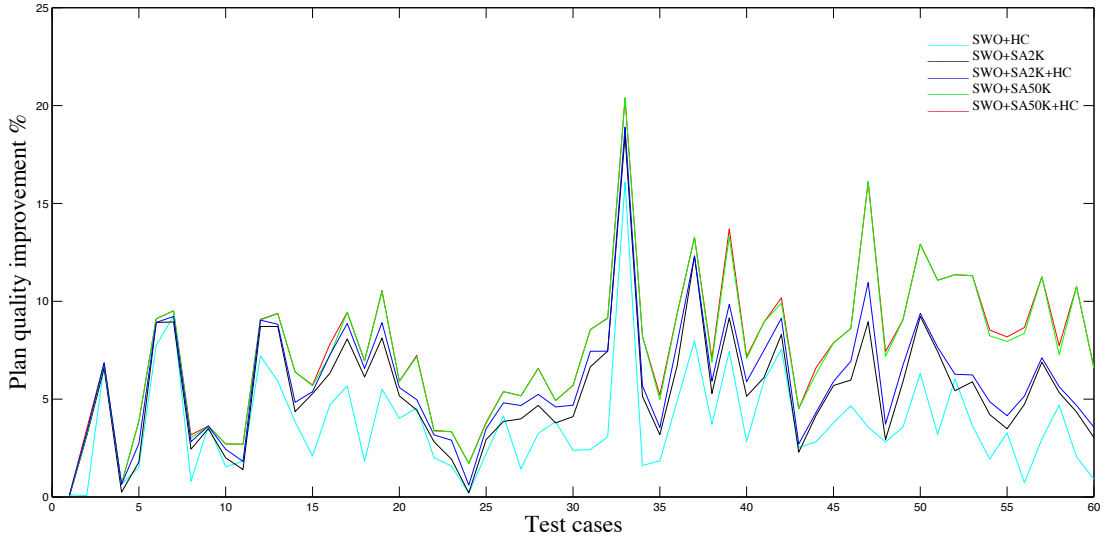


Fig. 4. Improvement of plan quality over SWO

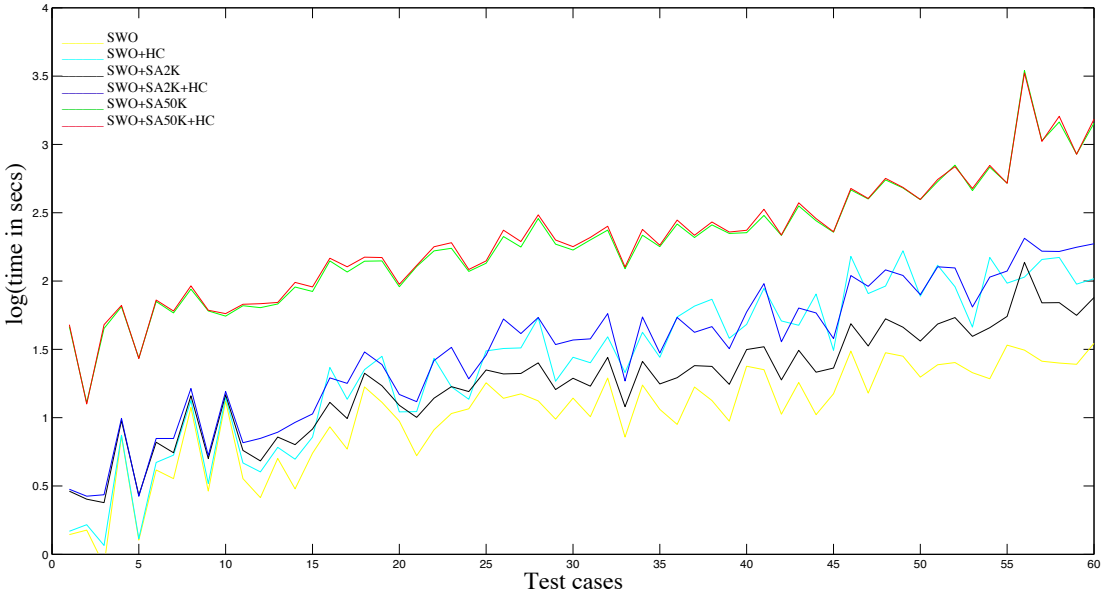


Fig. 5. Execution time results

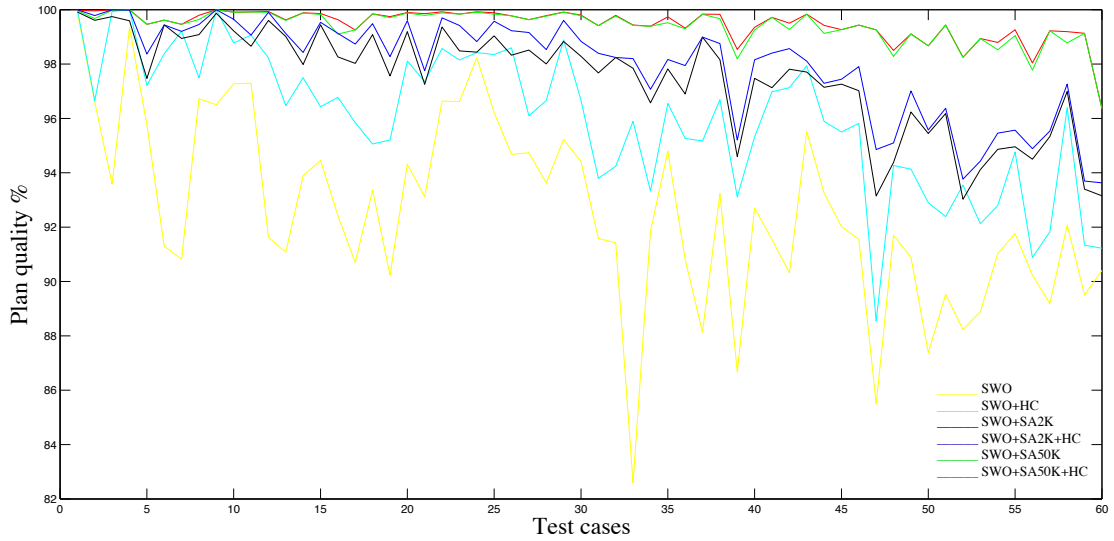


Fig. 6. Quality relative to the *experimental upper bound*

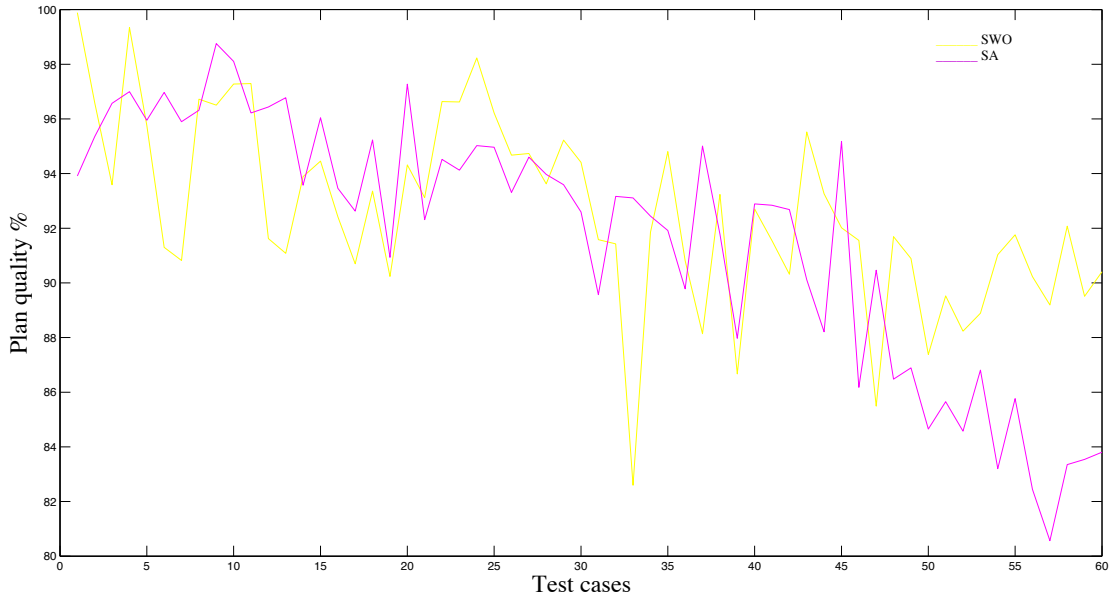


Fig. 7. Running SA from the empty plan for the same time as SWO. Quality relative to the *experimental upper bound*

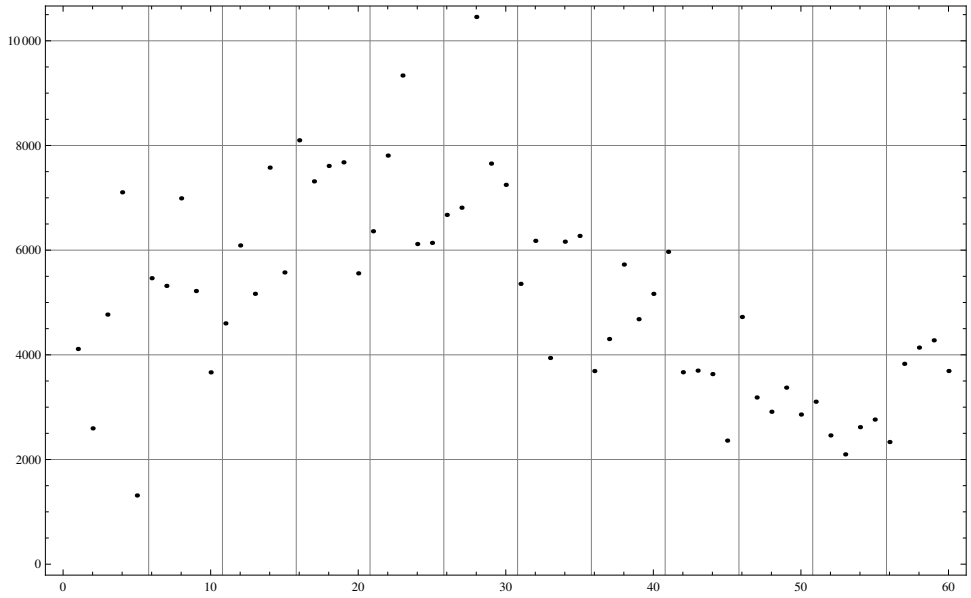


Fig. 8. Avarage neighborhood size for each test case

Finally, we ran SA as the only scheduler, that is, applying it to the empty plan, for all test cases. We did not use a predefined KMAX value for this test (as we did in the other experiments), but we adopted as a termination criterion the run time t_i^{SWO} spent by SWO on problem instance i, using the same settings as in the previous work [18]. The results, compared to the experimental upper bound, are shown in Figure 7. The average utility of the SWO runs is 92.58%, whereas the average utility of the SA runs is 91.72%. As it can be seen, SA is competitive to SWO for small problems, however SWO clearly wins on the hard problem instances. Thus, running first SWO and then SA in a post-processing phase is a dominant strategy compared to running SA from the beginning, when applied to hard problems.

Additional experiments have shown that by giving more time to SWO (compared to the settings used in previous work [18]) no improvement is noticed; thus, employing stochastic local search in a post-processing phase is the only option to further optimize these plans, provided that more time is available. After thorough investigation of SWO's behavior we realized that the main obstacle for SWO is that it searches in two coupled spaces, that is, the priority space and the solution space (Figure 1b). Not every solution in the solution space has a corresponding ordering of the activities in the priority space. So, many interesting solutions cannot be reached by pure SWO, thus why the need to couple SWO with a post-processing search algorithm.

On the Size of the Neighborhood

The average neighborhood size for the 60 problem instances was computed using the SWO+SA2K+HC configuration of the scheduler. Since SA does not compute all the neighbors of a solution, we used HC's *best_neighbor* function, which tries every transformation that can be applied to a solution. The reported values are averages over all iterations of HC, 10 times per problem.

The results are presented in Figure 8. The problem instances with 25 and 30 activities have the largest neighborhood size. For problems with fewer activities, the neighborhood size is smaller as expected. For problems with more activities within the same time horizon, the neighborhood size is again smaller due to the complex interactions between the activities that render many transformations invalid.

Contribution of Individual Transformations

Using SWO+SA2K as our basic configuration, we removed transformations from the post-processor, one of them at a time. In Figure 9 we compare each transformation's contribution, based on the transformations' absence from SWO+SA2K.

The top line (0%) represents the total utility of the solutions produced by SWO+SA2K, whereas for each transformation we see the percentage decrement from SWO+SA2K, when that transformation is omit-

ted from the post-processor. The results vary depending on the problem instance. For example, test cases #52 and #60 (hard problems) demonstrate a large drop in total utility when removing any one of the transformations, whereas in other cases removing a single transformation produces minor reduction in the total utility of the resulting solutions. These results of course depend also on the improvement that SA2K induced over the seed solution produced by SWO. The fact that by removing any one of the transformations we get a drop in the quality of the produced solutions demonstrates that the synergy of all transformation is significant in the post-processing phase. The transformation *Add Activity* was not always applicable, but when it was its removal gave us large drops in total utility, as should be expected.

In Figure 10 we compare a minimal version of the SWO+SA2K with just a single transformation employed. Each transformation was tested separately to obtain a view of its contribution when used by itself. The only transformation not tested was *Location Changes*, as it relies on its combination with other transformations to alter the utility of a schedule, and does not modify the total utility by itself. The baseline represents SWO.

Some transformations managed to increase the total utility without relying on synergies with others. The most significant seem to be *Best Start*, that is, moving activity parts in the schedule and altering their $U(time)$ utility bonus, and *Increase Duration by one*, that is, increasing an activity part's duration by one time unit. Other transformations did not provide any benefit when used alone. For example, *Merge Activity Parts* does not seem to do much by itself, there were few cases where it provided a contribution to the solution quality. Others, such as *Add Activity*, did not contribute anything by themselves in most test cases, but when they did they provided a huge boost in utility.

A Realistic Scenario

In this section we compare SWO+SA50K against SWO on a realistic scenario. The motivation is twofold: First, to demonstrate the effectiveness of post-processing on a realistic problem and, second, to visualize the improvement incurred by post-processing over the seed plan. The scenario comprises an over-constrained problem; however, such scenarios are typical of busy people, who are reasonably expected to be interested in using intelligent assistance in managing their activities. In the following paragraphs we first describe the scenario informally, then we compare the two algorithms and finally we discuss their relative performance.

The scenario has as follows: Ann is a university teacher, with a tight weekly schedule. In addition to her regular commitments (that is, teaching, student hours, etc), this particular week she has five additional activities. Her regular working hours are 09:00 to 19:00, Monday to Friday. On this Friday morning however she is traveling to a conference abroad, in order to present a technical paper.

Her first two activities concern preparing educational material for two of her classes (Module A and Module B). For each one of these activities Ann needs at least two hours, in case she will exploit as much as possible from last year's material, and at most six hours, if she is going to prepare entirely new material. The extra duration, for the preparation of new material, has a high utility for her. Ann prefers to work on these activities during the evenings. The activities are interruptible; whenever Ann works on anyone of them, she should not spend less than one hour or more than two hours on it.

The third activity concerns preparing slides for the conference presentation. She wants to spend at least three hours on this activity, whereas spending two more hours might result in a more attractive presentation and, consequently, more utility for herself. Preparing the slides is a task better accomplished in the morning, whereas again the activity can be scheduled in parts of between one and two hours. The deadline of course is before the scheduled talk at the conference.

The fourth activity concerns participating at a meeting with a colleague, requiring physical presence at a nearby location that is approximately one hour away. Her colleague expects Ann either on Monday evening (between 17:00–19:00) or Tuesday afternoon (between 12:00–15:00), depending on Ann's availability. The estimated duration of the meeting is between one and two hours, however this activity has a low utility for Ann, so she is not enthusiastic about spending more time for the meeting.

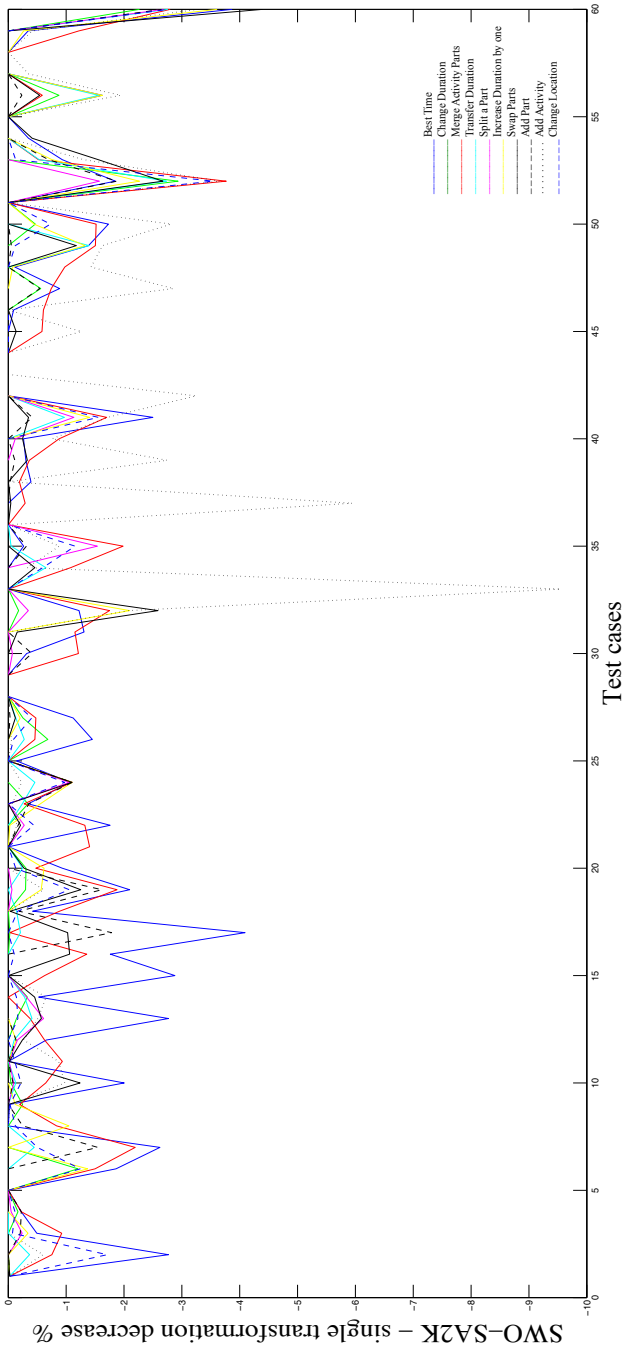


Fig. 9. Percentage decrease in the quality of the resulting plans when nine out of the 10 transformations are employed. Each line corresponds to removing a single transformation from their bundle.

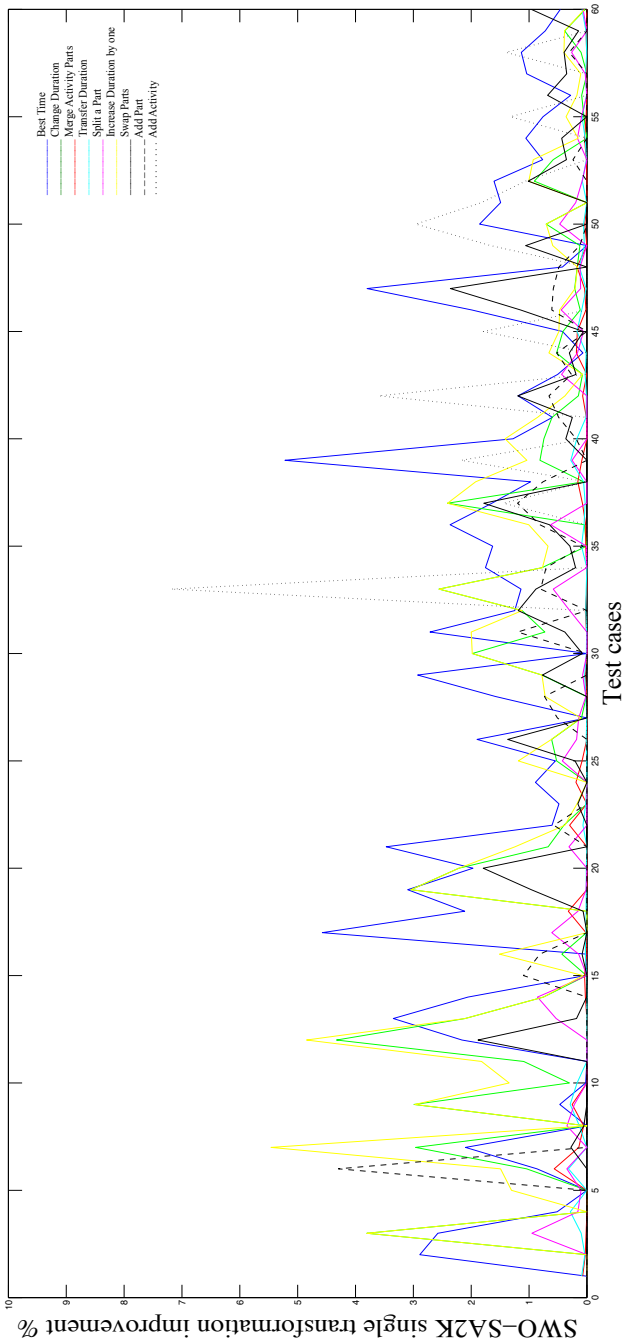


Fig. 10. Percentage increase in the quality of the resulting plans when each transformation is employed separately. Each line corresponds to a single transformation being employed.

The fifth activity concerns finishing and submitting an article for a journal special issue. This is a very important task for Ann, since she is working on the article for the last two months. She expects to spend at least three hours to finish the article; however, having another last read of the text would require six hours in total (high utility). Writing the article needs large compact periods of time, of at least one and a half hours and at most three hours. Since this activity requires her complete focus, Ann prefers to accomplish writing the article within a temporal period of at most 24 hours (low utility), whereas she also prefers to schedule it as early in a day as possible. She can only work on this activity though after the meeting with her colleague (ordering constraint), since the article concerns the project they are working on together.

Regular activities involve five lectures of two hours each, which Ann is giving to her classes, in a predetermined schedule; the regular one-hour weekly meeting with her PhD student, which can be scheduled between 13:00 and 18:00 on Thursday; a meeting with her colleagues at her Department (this is a fixed time commitment); weekly students hours, requiring two to three hours, that should be scheduled on Tuesday morning, between 9:00 and 13:00 (the utility of the extra duration is low); and, finally, daily afternoon lunches, to be scheduled between 15:00 and 17:00, having duration between half an hour and one full hour (again, the extra utility from having a long lunch is low).

The *loose upper bound* of the problem, that is, if all activities are scheduled with maximum duration and at their best possible intervals, was calculated to be $2,473.47$. A solution of that utility can easily be shown as unattainable, due to the over-constrained nature of the problem, as the available time is less that the sum of the maximum durations of all involved activities. Furthermore, there are conflicting temporal preferences, Ann needs some time to travel to the location of the fourth activity and come back, whereas there are additional constraints such as the fifth activity's maximum proximity preference. The *experimental upper bound* was found to be $2,082.2$. This was calculated by running SWO+SA50K+HC 1000 times on the problem.

SWO's solution has a utility of $1,831.53$, whereas SWO+SA50K's solution has a utility of $2,074.09$ (average over 100 runs), which is an improvement of 13.24% over SWO's. The two solutions are outlined in Figures 11 and 12. Since the post-processor is a stochastic algorithm, we executed it 100 times on Ann's problem and selected to present in Figure 12 the mode (most common) solution. The "meeting with colleague" activity is denoted with different color, because it is scheduled at a different location (that is, colleague's office) than Ann's university. The average time required by SWO+SA50K is 12 seconds on our system, which we consider acceptable. In contrast, SWO solved the problem in 0.1 seconds.

SWO (Figure 11) omitted two activities from the plan, that is "Module A preparation" and "Meeting with PhD student". The post-processor added these activities and returned a schedule with all seventeen activities included. Monday morning 9:00–10:00 was left unused by both algorithms, as the current time (that is, when the problem is solved) was considered to be Monday morning at 09:30 and there is no activity that can be broken into thirty minute parts and be scheduled at that time. Similarly, intervals 11:00–12:00 and 13:00–14:00 on Tuesday are used as the traveling time from Ann's university to her colleague's office and back. However, SWO resulted in three more unused half-hour intervals, on Monday, on Wednesday and on Thursday, that could not be used for any activity parts due to their short length.

Let's have a deeper look at the plan found by SWO. On Monday SWO placed two parts of "Prepare slides" next to each other, so the unused space was necessary as parts of the same activity must have at least 30 mins temporal distance to each other (proximity constraint). The unused space on Wednesday cannot be used by either "Prepare slides" or "Submit article" activities, since both adjacent activities are already scheduled with their maximum duration. The same is also true for the unused space on Thursday, since the lecture is a fixed-time commitment, whereas "submit article" is already scheduled at its maximum duration (at a total time of 6 hours).

Taking this schedule of Figure 11 as its seed solution, SWO+SA50K produced the schedule shown in Figure 12. There are no unused intervals in the improved schedule, whereas all activities are now scheduled, even if not with their maximum duration. On Monday there is a single part of "Prepare slides", thus making room for another part of "Module B preparation" and allowing for an one hour lunch. The total duration of "Prepare slides" has been decreased by an hour. On Tuesday, "Module B preparation" was decreased to one hour, and a part for "Module A preparation" was scheduled in the resulting free space. On Wednesday, the post-processor utilized the half-an-hour available duration out of the one-hour one it had gained for "Prepare slides" (by removing the one-hour

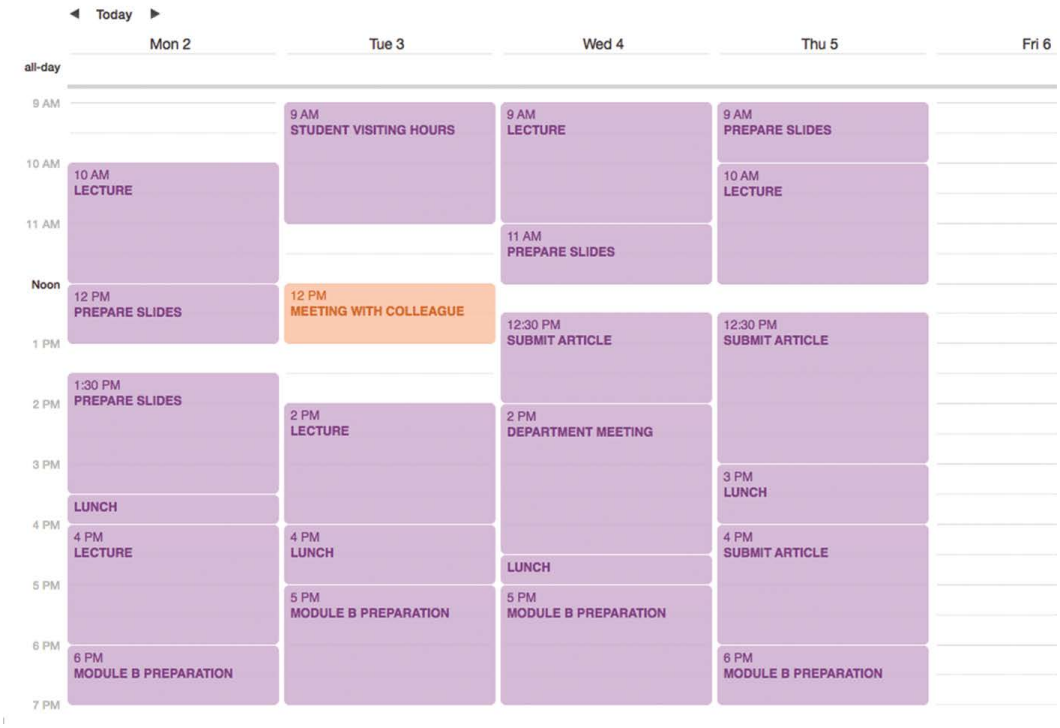


Fig. 11. SWO's solution

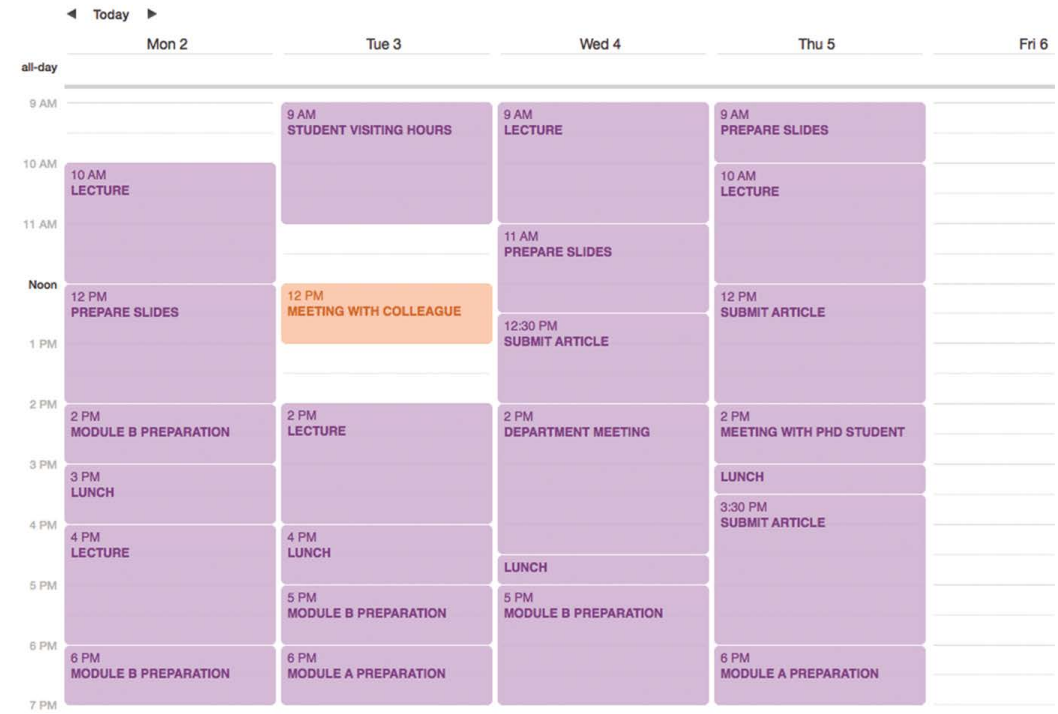


Fig. 12. SWO+SA50K's solution

part on Monday) to expand that activity to fill in the unused space. The rest of that day's schedule is unaltered. Finally, on Thursday, the two "Submit article" parts were swapped and the first one was moved earlier to fill in the unused space. The afternoon lunch was decreased to half-an-hour and the "Meeting with PhD student" was added in the resulting free space. At the 18:00–19:00 interval "Module B preparation" was replaced by "Module A preparation".

Activity "Finish article" gives the highest utility per unit of extra duration, so both schedules included it with its maximum duration (6 hours). In Figure 12 we can observe that the allocated duration for "Prepare slides" was decreased by half-an-hour compared to Figure 11, whereas the duration of "Module B preparation" was decreased by one hour. The one and a half hours freed, together with the one and a half hours of unused temporal space of Figure 11, were used for adding the two extra activities, that is, two hours for "Module A preparation" and one hour for "Meeting with PhD student".

One could argue that the post-processor should have freed one hour from "Prepare slides" and half-an-hour from "Module B preparation", as the extra duration for Module B is more important. But, the space was needed mostly for "Module A preparation", which has a higher utility for evening times, the same as module B, thus gaining utility from its temporal placement. Moreover "Meeting with PhD student" had a small available domain (could only be scheduled between 13:00–18:00 on Thursday with no other temporal preference).

It is important to emphasize that the solution presented in Figure 12 is not the best solution found by the post-optimization process; as we noted earlier, this is the most-common solution found over 100 runs, with a utility of $2,074.09$, whereas the best solution found had a utility of $2,082.2$. A user trying to manually improve the seed solution may occasionally find a better solution than the one suggested by the system. However, manually optimizing the seed solution could not be adopted as a regular practice, since it requires significant human effort, especially by inexperienced users. For example, by comparing the plans before and after applying post-optimization to the realistic scenario, we count more than ten activity parts that have been changed with at least one transformation. In this way we justify the need for intelligent solutions to produce optimized personal plans, such as the one presented in this article.

On the Contribution of Trimming and Backtracking in Over-Constrained Problems

Trimming and Backtracking are two techniques that are employed in over-constrained problems only. Since the realistic scenario is such a problem, it is interesting to have a closer view at their contribution (separately and jointly) to the outcome of the post-processing optimization. Neither backtracking nor trimming have any effect on the rest of the problems of our test suite, since even the tight problems with 60 activities are not over-constrained.

Table 1
SWO+SA50K 100 Runs

Backtracking	Trimming	Utility Average	St. Deviation
ON	ON	2,074.30	3.0863
OFF	ON	2,049.21	37.4489
ON	OFF	1,997.27	41.7054
OFF	OFF	1,861.80	17.0797

Table 1 presents aggregated results of the post-optimization phase over 100 runs on the realistic scenario, with four configurations: (a) with backtracking, with trimming, (b) without backtracking, with trimming, (c) with backtracking, without trimming, and (d) without backtracking, without trimming. As it is clear from the results, post-optimization in over-constrained problems, enhanced with trimming and backtracking, produces steadily better results with negligible deviation, compared to not using either trimming or backtracking or both. By having a closer look at the data, we notice that trimming is more important than backtracking, producing better average results with lower deviation. However, their combination is by far better, both in terms of utility, as (most significantly) in terms of deviation.

Discussion

The approach followed in this work can be applied to other Constraint Optimization Problems (COP) as well, particularly in project scheduling applications. The first step is to define in-domain characteristics of the problem, that affect the overall utility of a solution, and exploit them to produce a reasonable set of transformations to compute the neighborhood of any solution.

A transformation may be defined over a single decision variable, but can affect other decision variables as

well. A transformation may decrease the value of the objective function; however, this would allow subsequent transformations to increase it further. Transformations should be evaluated both in isolation, as well as in combination to each other. Transformations with no apparent and distinct contribution to the overall local search optimization process should be abandoned.

Hierarchical neighborhood exploration is another novel feature of the proposed work. In order to find a neighbor, first one major decision has to be made, such as what activity to alter. Then, the type of transformation has to be selected. Depending on the type, another decision might be necessary, i.e., which part of the activity to alter. Finally, all the valid values for the involved decision variable are generated and one of them is selected stochastically. Neither the number of the valid neighbors, nor its upper-bound, need to be known.

Hierarchical neighborhood can be combined with caching subsets of neighbors. The first time a specific combination of activity, transformation and part of activity is encountered, the set of valid values for the involved decision variable can be cached. Note that extracting the set of valid values for a specific decision variable is a time demanding process. So, the next time the same combination is encountered, without having accepted any neighbor in the between, valid neighbors can be retrieved from the cache, thus accelerating SA significantly, especially during the last stages of its execution, when accepting a neighbor occurs less often.

Summary and Future Work

In this article we employed local search optimization methods to further improve the quality of individual activity personal plans with complex constraints and preferences, that were originally produced by an adaptation of the Squeaky Wheel Optimization framework. Due to the complexity and interplay of the constraint optimization model, local search seems to be a necessity to obtain locally optimal plans.

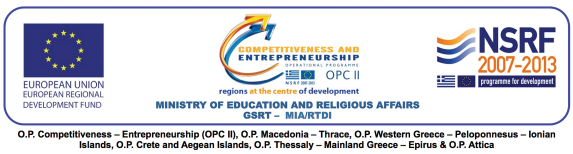
Based on extensive experimental results, we found two practical configurations of the proposed post-optimization modules: The first one, which is optimized for speed, consists of a simulated annealing post-processing phase with a predefined moderate number of iterations. The second one, which is optimized for utility, couples an extended simulated annealing phase with a hill-climbing one. Local search is based on a series of transformations over the schedules, resulting in a large neighborhood. As demonstrated in the article, all transformations, either independently or jointly, have a value in getting qualitative solutions. Furthermore, we showed that similar results can be achieved by employing local search techniques only, i.e., using the empty plan as the start point. A real-world scenario was also employed to demonstrate the effectiveness of the proposed local search methods, presenting the schedules before and after post-optimization in the familiar calendar form.

For the future, we are considering further enhancements to both the model and the scheduling algorithm. Concerning the model, we are working on supporting joint activities, that is, activities where many persons are involved. We are also considering to support preferences over alternative locations for each activity. Our ultimate goal is to convert the constraint optimization problem to a planning problem, with activities having preconditions and effects, thus allowing for intelligent agent assistants that proactively add tasks into a user's task list.

Concerning the algorithms, we are working on devising new transformations for exploring the local neighborhood, including pre-specified combinations of the transformations presented in this article. Using heuristics to select a subset of the transformations to consider might be of great significance, in order to keep the whole approach efficient.

Acknowledgements

The authors are supported for this work by the European Union and the Greek Ministry of Education, Lifelong Learning 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] 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'12), in conjunction with CPAIOR-2012, Nantes, France., May 2012.

[2] 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, 2013.

[3] 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.

[4] 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, July 2011.

[5] L. Chrpa, T. L. McCluskey, and H. Osborne. Optimizing plans through analysis of action dependencies and independencies. In L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, editors, *ICAPS*. AAAI, 2012.

[6] D. Curran, E. Freuder, and T. Jansen. Incremental evolution of local search heuristics. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 981–982, New York, NY, USA, 2010. ACM.

[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. AAAI Press, 2008.

[8] L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29 – 57, 1993.

[9] D. Joslin and D. P. Clements. Squeaky wheel optimization. *J. Artif. Intell. Res. (JAIR)*, 10:353–373, 1999.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] H.-J. Lee, S.-J. Cha, Y.-H. Yu, and G.-S. Jo. Large neighborhood search using constraint satisfaction techniques in vehicle routing problem. In Y. Gao and N. Japkowicz, editors, *Advances in Artificial Intelligence*, volume 5549 of *Lecture Notes in Computer Science*, pages 229–232. Springer Berlin / Heidelberg, 2009.

[12] K. L. Myers, P. Berry, J. Blythe, K. Conley, M. T. Gervasio, D. L. McGuinness, D. N. Morley, A. Pfeffer, M. E. Pollack, and M. Tambe. An intelligent personal assistant for task and time management. *AI Magazine*, 28(2):47–61, 2007.

[13] H. Nakhost and M. Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In R. I. Brafman, H. Geffner, J. Hoffmann, and H. A. Kautz, editors, *ICAPS*, pages 121–128. AAAI, 2010.

[14] L. Palen. Social, individual and technological issues for groupware calendar systems. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 17–24, New York, NY, USA, 1999. ACM.

[15] I. Refanidis. Managing personal tasks with time constraints and preferences. In M. S. Boddy, M. Fox, and S. Thiébaux, editors, *ICAPS*, pages 272–279. AAAI, 2007.

[16] I. Refanidis and A. Alexiadis. Deployment and evaluation of selfplanner, an automated individual task management system. *Computational Intelligence*, 27(1):41–59, 2011.

[17] I. Refanidis, C. Emmanouilidis, I. Sakellariou, A. Alexiadis, R.-A. Koutsiamanis, K. Agnantis, A. Tasidou, F. Kokkoras, and P. S. Efraimidis. myvisitplanner $^{\mathrm{gr}}$: Personalized itinerary planning system for tourism. In A. Likas, K. Blekas, and D. Kalles, editors, *SETN*, volume 8445 of *Lecture Notes in Computer Science*, pages 615–629. Springer, 2014.

[18] I. Refanidis and N. Yorke-Smith. A constraint-based approach to scheduling an individual's activities. *ACM TIST*, 1(2):12, 2010.

[19] U. Schöning. Comparing two stochastic local search algorithms for constraint satisfaction problems. In F. Ablayev and E. Mayr, editors, *Computer Science – Theory and Applications*, volume 6072 of *Lecture Notes in Computer Science*, pages 344–349. Springer Berlin / Heidelberg, 2010.

[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.