Post-Optimizing Individual Activity Plans through Local Search

Anastasios Alexiadis and **Ioannis Refanidis**
Department of Applied Informatics, University of Macedonia,
Egnatia 156, 54006, Thessaloniki, Greece.
talex@java.uom.gr, yrefanid@uom.gr

Abstract

Post-optimization through local search is known to be a powerful approach for complex optimization problems. In this paper we tackle the problem of optimizing individual activity plans, i.e., plans that concern activities that 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 paper 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 using either hill climbing or simulated annealing. We present several experiments that demonstrate an improvement on the utility of the produced plans, with respect to the seed solutions produced by SWO, of more than 6% on average, which in particular cases exceeds 20%. Of course, this improvement comes at the cost of extra time.

Introduction

Calendar applications and digital personal assistants are typically based on a series of fully specified and independent events. Each of these events is defined among others by a fixed start time, a duration (or end-time) and, potentially, a location. Furthermore, many systems also support tasks. These are individual commitments potentially having a deadline to be met (e.g., writing a paper or doing the week's shopping). Tasks are usually kept separately in task lists and are not characterized by a specific start time. In these systems, as soon as a task is dropped into the calendar, it is transformed to an event.

The need to develop intelligent automated systems for calendar management has been considered ambitious for at least three reasons. Ethnographic studies have shown that people tend to seek appropriate and contextualized assistance (Palen 1999). Moreover users would not be able to define the scheduling domain explicitly, as that would require special training for the specification of their preferences with a formal representation. Finally, the scheduler must be able to account for complex and subtle preferences and constraints. While tracking this problem is considered ambitious, intelligent assistance with time and task management has been a recognized target for AI (Myers et al. 2007) (Freed et al. 2008) (Refanidis 2007) (Refanidis and Alexiadis 2011) (Berry et al. 2011) (Bank et al. 2012).

In (Refanidis and Yorke-Smith 2010) a model is presented to treat events and tasks, denoted as activities, in a uniform way. The model supports a number of unary and binary constraints and preferences over activities. Particularly, each activity is characterized by a temporal domain, a duration range, a set of alternative locations, interruptibility, utilization, preferences over the temporal domain and the alternative durations, constraints and preferences over the way parts of an interruptible activity are scheduled in time. The model also supports binary (ordering, proximity and implication) constraints and preferences between pairs of activities. 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.

In this paper we present local-search techniques that increase the quality of SWO's plan output through post-optimization. The use of local-search methods for constraint satisfaction problems has been done before (Curran, Freuder, and Jansen 2010) (Lee et al. 2009) (Schöning 2010) (Nakhost, Hoffmann, and Müller 2010). In our work, we devised and implemented a set of transformations of valid plans, such as shifting activities, changing their durations or locations, as well as merging or splitting parts of interruptible activities. Extensive experimental results have shown constant improvement over SWO's output up to 22.7%.

This paper extends previous work (Alexiadis and Refanidis 2012), by providing additional post-processing transformations to explore the neighborhood, enhanced with a stochastic local search algorithm (that is, *simulated annealing*) to avoid local maxima. As it is shown experimentally, additional improvement on the quality of the resulting plans is achieved with the new features of the post-processing phase.

The rest of the paper is structured as follows. First, we formulate the optimization problem and illustrate the SWO-based approach. Next, we present the local-search transformations to explore the neighborhood during the post-

optimization phase, using either *hill-climbing* or *simulating annealing* (Kirkpatrick, Gelatt, and Vecchi 1983) (Ingber 1993). Subsequently, we present experimental results over a large set of problem instances. Finally, we conclude the paper and identify directions of future work.

Background

In this section we present the problem formulation, as well as the SWO approach to cope with the problem.

Problem Formulation

In previous work (Refanidis and Yorke-Smith 2010), 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 given. For each activity $T_i \in T$, its minimum duration is denoted with d_i^{min} and its maximum duration with d_i^{max}. The decision variable p_i denotes the number of parts in which the *i-th* activity has been split, with $p_i \geq 1$. T_{ij} denotes the *j-th* part of the *i-th* activity, $1 \leq j \leq p_i$. The sum of the durations of all parts of an activity must be at least d_i^{min} and no greater than d_i^{max}.[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, must equal d_i.[2] Non-interruptible activities are scheduled as one part.

For each T_i, we define the minimum and maximum part duration $smin_i$ and $smax_i$,[3] as well as the minimum and maximum temporal distances between every pair of parts, $dmin_i$[4] and $dmax_i$.[5]

For each activity T_i, its temporal domain is defined as a set of temporal intervals defining $D_i = [a_{i1}, b_{i1}] \cup [a_{i2}, b_{i2}] \cup \ldots \cup [a_{i,F_i}, b_{i,F_i}]$, where F_i is the number of intervals of D_i.[6]

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 $l_{ij} \in Loc_i$[7] denotes the particular location where T_{ij} is scheduled.[8]

Activities may overlap in time. Each activity T_i is characterized by a utilization value, $utilization_i$.[9] At any time point, the set of activities that have been scheduled should have compatible locations (i.e., locations with no temporal distance to each other) and the sum of their utilization values should not exceed the unit.

The model supports four types of binary constraints: 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.[10] 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.[11] 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.[12]

Scheduling personal activities is considered a constraint optimization problem. That said, the empty schedule is a valid schedule but with low utility, thus we are interested in better schedules. There are several sources of utility. 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 linear and stepwise utility functions of time over the temporal domain of each activity.

Any form of hard constraint can also be considered a soft constraint that might contribute utility. So, minimum and maximum distance constraints between the parts of an interruptible activity might contribute $U_{dmin_i}(\pi_i)$ and $U_{dmax_i}(\pi_i)$ respectively. 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 for a partial preference p, denoted with $DoS(p)$, is defined as the ratio of the number of infinitesimal pairs of parts, one from T_i and another from T_j, for which the binary preference holds, to the total number of infinitesimal pairs of 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.

---

[1] $\forall T_i,\ d_i^{min} \leq d_i \leq d_i^{max}$ OR $d_i = 0$ $(C1)$

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

[3] $\forall T_{ij}, smin_i \leq d_{ij} \leq smax_i$ $(C3)$

[4] $\forall T_{ij}, T_{ik}\ j \neq k \Rightarrow t_{ij} + d_{ij} + dmin_i \leq t_{ik} \vee t_{ik} + d_{ik} + dmin_i \leq t_{ij}$ $(C4)$

[5] $\forall T_{ij}, T_{ik}\ j \neq k \Rightarrow t_{ij} + dmax_i \geq t_{ik} + d_{ik} \wedge t_{ik} + dmax_i \geq t_{ij} + d_{ij}$ $(C5)$

[6] $\forall T_{ij}, \exists k, 1 \leq k \leq F_i : a_{ik} \leq t_{ij} \leq b_{ik} - d_{ij}$ $(C6)$

[7] $l_{ij} \in Loc_i$ $(C7)$

[8] $\forall T_{ij}, T_{mn}, T_{ij} \neq T_{mn} \wedge (Dist(l_{ij}, l_{mn}) > 0 \vee Dist(l_{mn}, l_{ij}) > 0) \Rightarrow t_{ij} + d_{ij} + Dist(l_{ij}, l_{mn}) \leq t_{mn} \vee t_{mn} + d_{mn} + Dist(l_{mn}, l_{ij}) \leq t_{ij}$ $(C8)$

[9] $\forall t, \sum_{T_{ij}:\ t_{ij} \leq t < t_{ij} + d_{ij}} utilization \leq 1$ $(C9)$

[10] $\forall T_i, T_j, T_i < T_j \Leftrightarrow d_i > 0 \wedge d_j > 0 \Rightarrow \forall T_{ik}, T_{jl}, t_{ik} + d_{ik} \leq t_{jl}$ $(C10)$

[11] $\forall T_{ik}, T_{jl}, t_{ik} + d_{ik} + dmin_{ij} \leq t_{jl} \vee t_{jl} + d_{jl} + dmin_{ij} \leq t_{ik}$ $(C11)$

$\forall T_{ik}, T_{jl}, t_{ik} + dmax_{ij} \geq t_{jl} + d_{jl} \wedge t_{jl} + dmax_{ij} \geq t_{ik} + d_{ik}$ $(C12)$

[12] $\forall T_i, T_j, T_i \Rightarrow T_j \Leftrightarrow d_i > 0 \Rightarrow d_j > 0$ $(C13)$

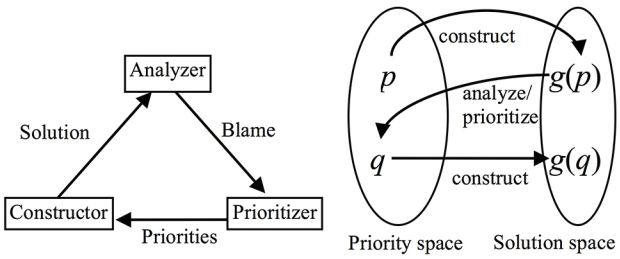

Figure 1: (a) The SWO cycle. (b) Coupled search spaces

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 start times t_{ij}, their durations d_{ij} and their locations l_{ij}, while trying to maximize the following objective function:

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

subject to constraints (C1) to (C13).

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

The SWO Approach

In (Refanidis and Yorke-Smith 2010), the problem is solved using the Squeaky Wheel Optimization (SWO) framework (Joslin and Clements 1999). 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 tasks determined by a priority queue. The solution is then analyzed to obtain the tasks that cannot be scheduled. Their priorities are increased, enabling the constructor to deal with them earlier on the next iteration. The cycle will be repeated till 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 tasks 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 ones on the solution that is generated.

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 (Selman, Kautz, and Cohen 1995). 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 t_{ij}, d_{ij} and l_{ij}, for each T_{ij}, while trying to maximize Formula (1).

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

Figure 2: Hill-climbing based post-processing algorithm

Applying Local-Search Methods to Enhance SWO Output

We applied *hill-climbing* (HC) using the output of SWO as the seed value, to further enhance the solution quality. The overview of the approach is presented in Figure 2.

U denotes the objective function of formula (1) and Solution is a complete and valid assignment of values to the decision variables t_{ij}, d_{ij} and l_{ij}, for each T_{ij}. HC is initialized with the problem definition (*Activities*) and SWO's output solution as the initial *Best_Solution*.

Calculating the Best Neighbor

The *best_neighbour* function computes neighbor solutions by attempting various changes to the decision variables of the parts of activities. The algorithm always keeps the solution resulting from the change that produces the highest utility. When all the available changes have been attempted on the decision variables of every part, the algorithm chooses the neighbor solution with the best utility to continue.

For every part T_{ij} of each activity T_i, the following transformations are attempted:

Best Start Time: This transformation attempts different values for the decision variable t_{ij} of part T_{ij}, one decision variable at a time. The values attempted always belong to the temporal domain of activity T_i. For each value attempted, the constraints are checked to examine if the change is consistent with them. If it is not, it is ignored.

Changing the duration: This transformation attempts different values for the decision variable d_{ij} of part T_{ij}. The values d_{ij} are always between $smin_i$ and $smax_i$, as defined for activity T_i. Every change is checked for constraint consistency.

Decreasing the duration of a task will not result in improving the overall quality; however, if it is combined with a stochastic local search procedure like simulated annealing, interesting results arise.

Merging two parts of an activity: This transformation attempts to merge two parts of an activity into a single part. For part T_{ij}, it iterates over all other parts of activity T_i, and for every T_{ik}, $j{\neq}k$, it attempts to remove T_{ik} and move its duration to T_{ij}. T_{ik} is then removed.

For every T_{ik}, where $j{\neq}k$, the function first checks whether $d_{ij}{+}d_{ik}{\leq}smax_i$ holds. If it does not, it ignores T_{ik}. Otherwise it attempts to produce two new solutions. In the former solution, d_{ij} is increased by d_{ik}; in the latter, d_{ij} is increased by d_{ik} and t_{ij} is decreased by d_{ik}. In other words, the duration of the removed part is added either at the end or at the beginning of part T_{ij}. As above, every solution produced is checked for constraint consistency.

Transfering duration between parts of the same activity: This transformation attempts to transfer duration between two parts of the same activity. For part T_{ij}, it iterates over all other parts of activity T_i, and for every T_{ik} where $j{\neq}k$ it attempts to transfer duration from T_{ik} to T_{ij}.

For every T_{ik} (where $j{\neq}k$, $d_{ik}{>}smin_i$ and $d_{ij}{<}smax_i$), up to four neighboring solutions are computed. These are the following:

1. Move the maximum allowed duration ($trans\_d$) by setting $d_{ik} = d_{ik} - trans\_d$ and $d_{ij} = d_{ij} + trans\_d$. This moves $trans\_d$ duration from the end of T_{ik} to the end of T_{ij}.
2. Move the maximum allowed duration ($trans\_d$) by setting $d_{ik} = d_{ik} - trans\_d$, $d_{ij} = d_{ij} + trans\_d$ and $t_{ij} = t_{ij} - trans\_d$. This moves $trans\_d$ duration from the end of T_{ik} to the beginning of T_{ij}.
3. Move the maximum allowed duration ($trans\_d$) by setting $d_{ik} = d_{ik} - trans\_d$, $d_{ij} = d_{ij} + trans\_d$, $t_{ij} = t_{ij} - trans\_d$ and $t_{ik} = t_{ik} + trans\_d$. This moves $trans\_d$ duration from the beginning of T_{ik} to the beginning of T_{ij}.
4. Move the maximum allowed duration ($trans\_d$) by setting $d_{ik} = d_{ik} - trans\_d$, $d_{ij} = d_{ij} + trans\_d$ and $t_{ik} = t_{ik} + trans\_d$. This moves $trans\_d$ duration from the beginning of T_{ik} to the end of T_{ij}.

The maximum allowed duration, $trans\_d$, is calculated as $min(smax_i - d_{ij}, d_{ik} - smin_i)$. If the constraint consistency check on all the four new solutions fails, $trans\_d$ is decreased by one. This will proceed until either a new valid solution is created or $trans\_d = 0$.

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 $d_{ik} = smin_i$, $l_{ik} = l_{ij}$ and the function will search for a suitable t_{ik} that succeeds on the consistency check of the new solution. If no such valid t_{ik} is found, nothing is computed.

Increasing the duration of an activity: This transformation attempts to increase the duration of a part and, consequently, of the whole activity. For part T_{ij}, where $d_{ij}{<}smax_i$, it attempts to produce up to two solutions. First, $d_{ij}+1$ is attempted. This will increase the part's duration by 1 and place the extra duration at the end of the part. On the second case, $d_{ij} + 1$ and $t_{ij} - 1$ are attempted. This will increase its duration by 1, by putting the extra duration the beginning of T_{ij}.

Swapping parts of different activities: This transformation attempts to swap two parts' start times between different activities. So, for each part T_{ij} of activity T_i, it is attempted to exchange t_{ij} with each t_{kx} of the *x-th* part of *k-th* activity.

Adding a part: This function is different from the previous ones in that it does not operate on parts but on activities, by attempting to add new parts. For an activity T_i, where:

$$\sum_{j=1}^{p_i} d_{ij} \leq (d_i^{max} - smin_i) \tag{2}$$

it is attempted to create a new part T_{ik} where $k = p_i{+}1$. The new part will have $d_{ik} = smin_i$, $l_{ik} = l_{ij}$(where $j = p_i$) and the function will search for a suitable t_{ik} that succeeds on the consistency check of the new solution. If no such valid t_{ik} is found, nothing is computed.

Adding an Activity: Similar to the previous transformation, this one is only applied on activities. If an activity is found, that was not scheduled for any reason (such as constraint violations with other activities of greater utility etc), this—and only this—transformation will be applied to that activity, as the other transformations are not defined on activities that do not have any scheduled parts. The purpose of this transformation is to include activities in the plan, that were not scheduled, by taking advantage of plan changes from the other transformations.

For an activity T_i that is not included in the current schedule, this transformation aggressively tries to schedule it by inserting parts to the timeline and within the domain of the task, 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 minimum possible duration and at the first found possible location. Due to the aggressive policy of this transformation, it might be the case that an activity will not be inserted in the current schedule, although this was possible.

Changing Locations: For every part T_{ij}, the *best_neighbour* transformation attempts to change its location and then to apply all the above methods described, except *Adding a Part* and *Adding an Activity*, which refer to activities. For every possible location $l_i \in Loc_i$ of T_{ij}, the *total travelling time* of the resulting solution is computed. Traveling time between two consequent parts, T_{ij} and T_{ul}, scheduled at l_{ij} and l_{ul}, is defined as $Dist(l_{ij}, l_{iu})$. If a suitable value for l_{ij} is found that reduces the *total travelling time*, all the above methods are re-applied on the resulting solution. The location change is then reset when we move to the next part.

This transformation allows to explore combinations of location changes with the above functions described. The logic behind the method is to search for any potential location changes that decrease the *total travelling time* of the schedule, thus giving the post-optimization algorithm more rescheduling options.

Avoiding Local Maximums

The *hill-climbing* post-processing algorithm can get stuck in local maxima. Algorithms based on *simulated annealing* (Kirkpatrick, Gelatt, and Vecchi 1983) are another type of local-search algorithms that can be applied to the above problem. Hill-climbing never makes a descending move (to states of lower global utility) and thus is not complete. In contrast a pure *random walk* is complete but extremely inefficient.

As a middle-ground to the above two extremes, we replaced hill-climbing as a post-processing algorithm with one based on *simulated annealing*. *Annealing* is the process in which one starts at a high energy state and high temperature levels, attempting to minimize the energy gradually—while lowering the temperature as well—to reach a low-energy state. Lower temperatures decrease the chances of moves to higher-energy levels (moves to worse states than the current one).

Low-energy states are reversed in our model to describe *high* global utility solutions. As a result, the simulated annealing based post-processing algorithm, presented in Figure 3, starts from the solution that results from the preceding SWO phase and gradually increases its utility. In addition, we empowered *simulated annealing* with *tabu lists*, to avoid returning to previously visited solutions. Finally, we converted all the *transformation functions* of the previous section to return all the valid neighbor solutions they calculated, instead of the best one.

There is a high probability of descending moves (to lower utility solutions), at the start of execution, which gradually decreases as the temperature variable decreases as well. This feature allows us to expand the search-space and avoid local maxima, whereas the gradual decrement allows the search algorithm to converge to a high-utility solution.

In the simulated annealing based post-processing algorithm we use the *random_neighbour* function (instead of *best_neighbour*), which returns a random neighbor solution. For increased efficiency, we define the variable lists $C_{ikt} \in C$, where each one of them holds all the valid neighbor solutions for an application of the t *transformation function*, for an activity part T_{ik} or a whole activity T_i (in the case of *Adding a Part* or *Adding an Activity*). For activities without any scheduled parts, only *Adding an Activity* is used (which is not used for any scheduled ones). Each C_{ikt} is created dynamically when required. As an example, if the transformation *Transfer Duration* (with no location changes) is chosen for the activity part T_{ik}, the search algorithm will pop a solution out of $C_{ikTransferDuration}$. If the previous list does not exist, the transformation will be computed on the above part and $C_{ikTransferDuration}$ will be created—with all the valid neighbor solutions found. Whenever the *random_neighbour* function chooses a solution from a transformation on a part, that has already been computed, the only overhead for the algorithm will be to obtain that solution from the list. When the current solution changes, the above lists are deleted and will be recomputed on demand. The number of valid solutions, for a transformation applied on a part, is not known beforehand, thus creating the need for the above variables.

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

```
if K ≥ KMAX  OR  U(Best) ≥ EMAX
  return Best
NTabu ← Tabu ∪ Solution
T ← SCHEDULE[K]
C ← ∅
DO i ← 0 to ∞
  if i+ K = KMAX
      return Best
  New ← random_neighbour(Activities,Solution,C),
New ∉ NTabu, (C_ikt ∈ C) ← (C_ikt ∈ C) \ New
  ΔE ← U(New) − U(Solution)
UNTIL ΔE > 0  OR  select New with probability e^(ΔE/T)
New_K ← K + i + 1
if New > Best
  New_Best ← New
else New_Best ← Best
return enhanced-post-process-swo(Activities, New,
New_Best, New_K, NTabu, SCHEDULE, KMAX, EMAX)
end
```

Figure 3: Simulated annealing based post-processing algorithm

The search algorithm uses the following parameters: SCHEDULE, KMAX, EMAX and $|Tabu|$. SCHEDULE defines the cooling schedule of the *simulated annealing* search algorithm. We used a cooling schedule dependent on KMAX that is defined as: SCHEDULE[K] = SCHEDULE[K-1] $\times$ $(1 - 0.07 \times \frac{100}{\text{KMAX}})$, where SCHEDULE[1] = 0.9. KMAX (the number of iterations of the simulated annealing algorithm) was set to 2000 and EMAX (the upper bound of the problem) to the upper bound of the particular problem instance being solved. We set $|Tabu|$ (the number of past solutions kept in the tabu list) to $\frac{\text{KMAX}}{10}$.

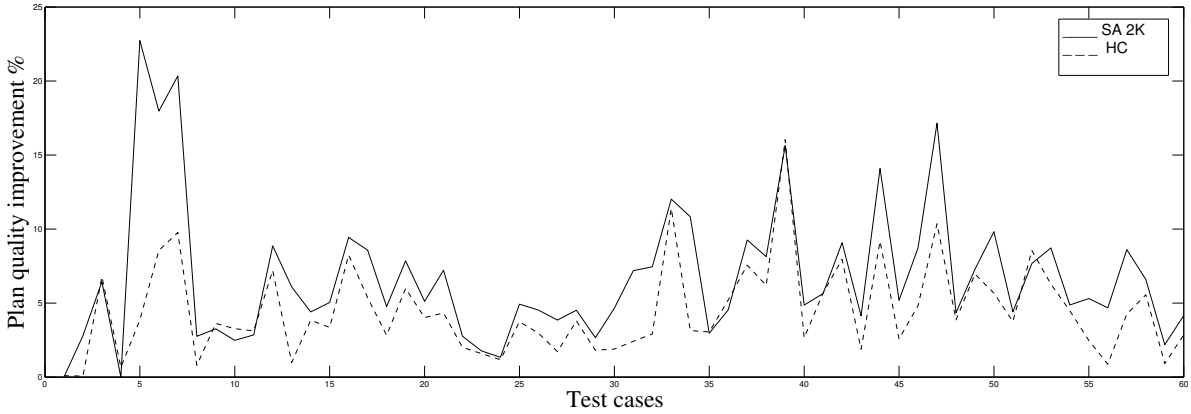

Figure 4: Improvement of plan quality over SWO. Dashed line concerns hill-climbing (HC), whereas solid line concerns simulated annealing (SA 2K) with 2,000 iterations.

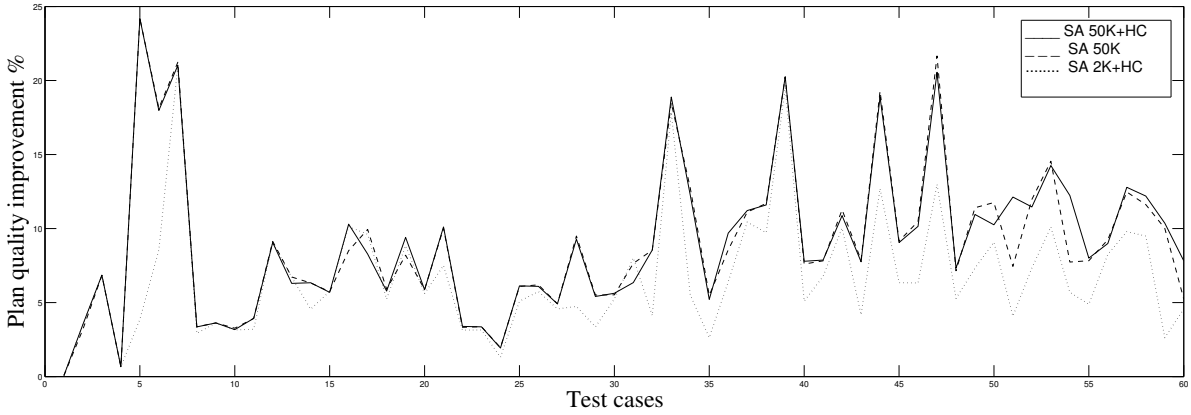

Figure 5: Improvement of plan quality over SWO for various configurations of the simulated annealing based post-processing phase. SA 50K refers to SA with 50,000 iterations. SA 50K+HC refers to SA 50K coupled with hill climbing. SA 2K+HC refers to SA with 2,000 iterations, coupled with HC.

Evaluation

We first compared the original SWO algorithm versus SWO coupled with the hill-climbing post-processing algorithm, as well as versus the SWO coupled with the simulated annealing one, on 60 test cases, ranging in size, taken from (Refanidis and Yorke-Smith 2010). The implementation of the above algorithms was done in C++ and the experiments were run on an Intel Xeon 2.66GHz processor.

The results of the first comparison are shown in Figure 4. The plot represents the plan quality percentage improvement of the two post-processing algorithms to the original SWO. The dashed line concerns the hill-climbing (HC) post-processing algorithm, whereas the solid line represents the simulated annealing post-processing algorithm with 2,000 iterations (SA 2K).

The test set (which was created for benchmarking SWO's solution quality and speed of execution) consists of five problems per number of activities, ranging from five activities to sixty, in steps of five. All the problems include activities with large temporal domains with many intervals each. A number of binary constraints and preferences were defined on them.

As we can observe from Figure 4, there is an improvement in all test cases with both HC and SA 2K. The best case was a 22.7% improvement in plan quality with SA 2K. The average was a an improvement of 4.6% for HC and 6.7% for SA 2K. As can be seen from the above results, SA 2K performed better than HC. Moreover, we can observe that SWO does not reach local maxima, though it approaches them in a satisfactory manner.

Next, we run some tests with more aggressive parameters, which are displayed in Figure 5. The dotted line represents a combination of the two algorithms, i.e. hill-climbing running on top of the resulting solution of SA (SA 2K+HC),

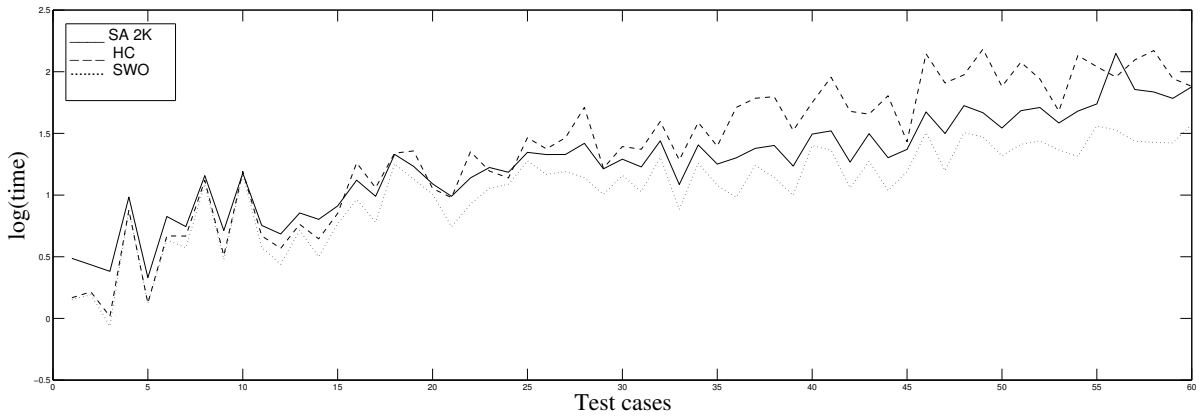

Figure 6: Execution time results for SWO, HC and SA 2K.

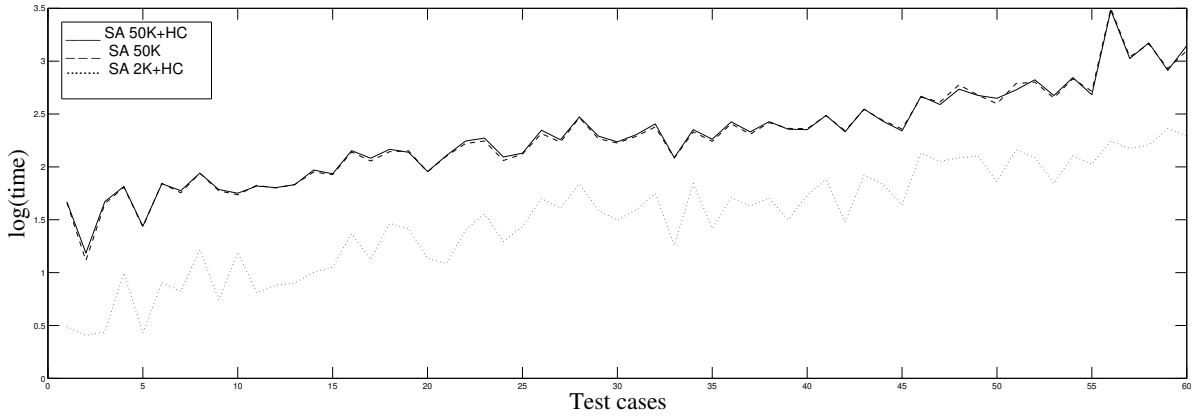

Figure 7: Execution time results for SA 50K, SA 50K+HC and SA 2K+HC.

in order to reach the nearest local maximum. The improvement over SWO was 7.03% on average. The solid line represents the combination of the two algorithms again, with a more aggressive KMAX$= 50000$ for simulated annealing (SA 50K+HC). The average improvement was 9.8%, with a best case of 24.19%. Finally, the dashed line represents the results of simulated annealing only, with KMAX$= 50000$ (SA 50K). The average improvement was 9.54% with a best case of 24.19%.

The percentage of time penalty was greater on the larger problems than the ones with fewer activities. Figure 6 compares the execution time of standard SWO to HC and SA 2K. The times given for HC and SA 2K are the combined values of the time of the main scheduler plus the post-processor. The figure is in logarithmic (base 10) scale. The dotted line represents SWO, dashed represents HC and solid represents SA 2K. The worst cases were encountered with the larger problem instances. On instances with up to 20 activities—which is the usual case for practical problems—the execution time of the three algorithms remains close. On the larger problem instances, SA 2K outperforms HC in terms of execution time, whereas simultaneously it provides better quality results as 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 fewer activities with less interdependencies between them.

In Figure 7 we compare the execution time of the more aggressive algorithms, SA 50K, SA 2K+HC and SA 50K+HC (combined values of scheduler plus post-processor). SA 50K and SA 50K+HC were the slowest, though their execution time is close, as the cost of the HC phase is overshadowed by the execution time of the SA with the large KMAX value. On the other hand, SA 2K+HC is 114% slower than SA 2K on average.

Finally, we compared the performance of the above algorithms to the *loose upper bound* of their respective problem

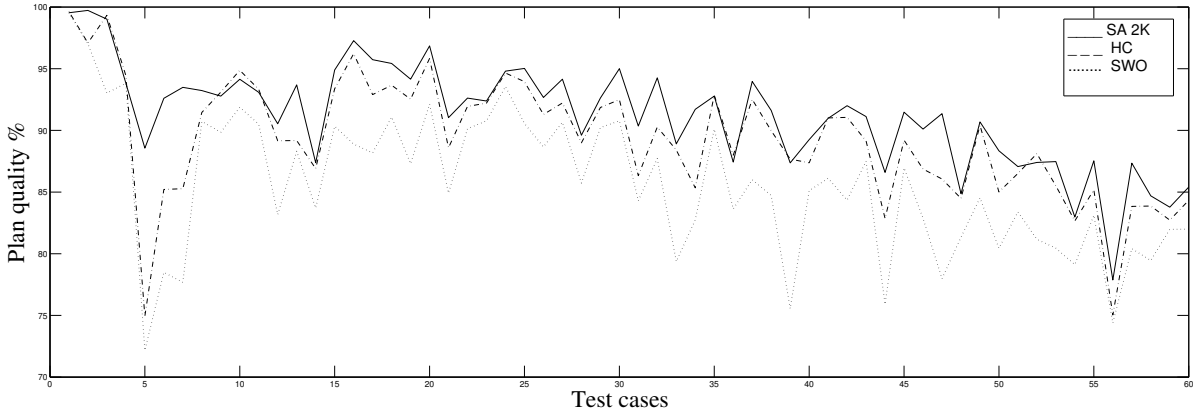

Figure 8: Quality relative to the *upper bound* for SWO, HC and SA 2K.

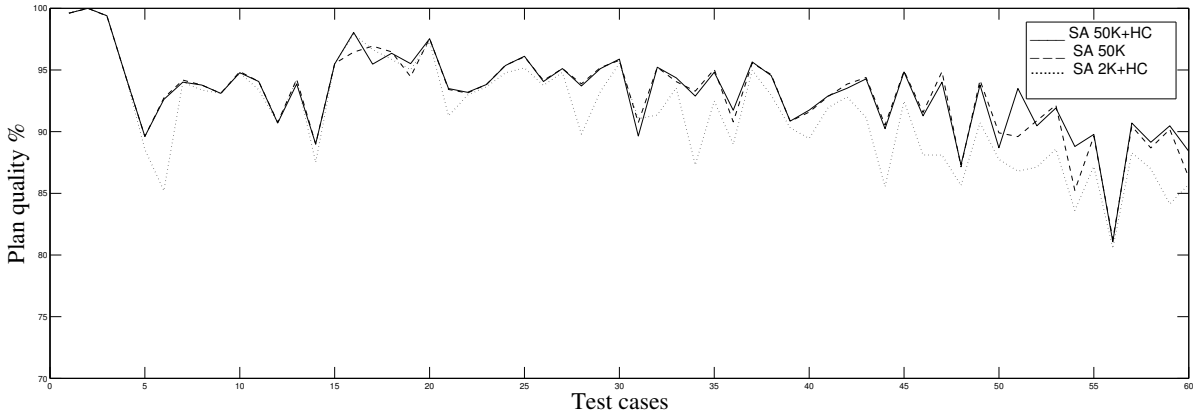

Figure 9: Quality relative to the *upper bound* for SA 50K, SA 50K+HC and SA 2K+HC.

instances. These results are shown in Figures 8 and 9. Particularly, SWO had an average plan quality of 85% to the loose upper bound, whereas HC and SA 2K had 89% and 91% respectively. Coupling SA with HC resulted in less than 1% of average improvement relative to SA, whereas increasing KMAX raised it to 93%.

The worst case for SWO was a plan quality of 72%, which was increased to 75% with HC, and to 89% with SA 2K. SA 50K+HC further raised it to 90%.

Conclusions and Future Work

In this paper we employed local search post-optimization methods to further improve the quality of personal plans with complex preferences, that were originally produced by an adaption of the Squeaky Wheel Optimization framework. Due to the complexity and interplay of the preference model, post processing seems to be a necessity to obtain locally optimal plans.

Based on our experimental results, we found two practical configurations of the proposed post-optimization modules: The first one, which is optimized for speed of execution, consists of a simulated annealing post-processing phase with a predefined moderate number of iterations. The second one, which is optimized for quality of solutions (producing high-utility local optimal solutions), couples a long simulated annealing phase with a hill-climbing one.

For the future, we are considering new enhancements to both the model and the scheduling algorithm. Concerning the model, we are working on the inclusion of joint activities, i.e., activities where many persons are involved, and on non-monotonic temporal domain preferences—for greater flexibility when specifying the user's temporal preferences. Another option that is considered is to convert the problem definition to a planning problem, with preconditions and effects, which will allow the model to be greatly enriched.

Concerning post-optimization, we are working on devising new transformations for exploring the local neighborhood, including selected combinations of the transformations presented in this paper. Using heuristics to select a subset of the transformations to consider might be of great

significance, in order to keep the whole approach efficient. Another thought we are considering is comparing the post-processing module applied on a random initial solution to SWO+SA. Moreover a formal analysis of real-world test cases would enable us to produce test data more closely related to real-world problem instances. Last of all, we will compare the individual transformations' contribution to each other in producing the final solution.

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

Alexiadis, A., and Refanidis, I. 2012. 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.

Bank, J.; Cain, Z.; Shoham, Y.; Suen, C.; and Ariely, D. 2012. 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, 2667–2672. New York, NY, USA: ACM.

Berry, P. M.; Gervasio, M.; Peintner, B.; and Yorke-Smith, N. 2011. Ptime: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.* 2(4):40:1–40:22.

Curran, D.; Freuder, E.; and Jansen, T. 2010. Incremental evolution of local search heuristics. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, 981–982. New York, NY, USA: ACM.

Freed, M.; Carbonell, J.; Gordon, G.; Hayes, J.; Myers, B.; Siewiorek, D.; Smith, S.; Steinfeld, A.; and Tomasic, A. 2008. Radar: a personal assistant that learns to reduce email overload. In *Proceedings of the 23rd National Conference on Artificial intelligence - Volume 3*, AAAI'08, 1287–1293. AAAI Press.

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

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

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

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

Myers, K.; Berry, P.; Blythe, J.; Conley, K.; Gervasio, M.; McGuinness, D.; Morley, D.; Pfeffer, A.; Pollack, M.; and Tambe, M. 2007. An intelligent personal assistant for task and time management.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2010. Improving local search for resource-constrained planning. In Felner, A., and Sturtevant, N. R., eds., *SOCS*. AAAI Press.

Palen, L. 1999. 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, 17–24. New York, NY, USA: ACM.

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

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

Refanidis, I. 2007. Managing personal tasks with time constraints and preferences. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *ICAPS*, 272–279. AAAI.

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

Selman, B.; Kautz, H.; and Cohen, B. 1995. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 521–532.