Planning Individual Activities through an Intelligent Calendar

by

Anastasios Alexiadis

B.Sc. Computer Science, Brunel University (2003)

M.Sc. Intelligent Systems, University of Sunderland (2005)

M.Sc. Applied Informatics, University of Macedonia (2009)

Submitted to the Department of Applied Informatics
School of Information Sciences
In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Applied Informatics

at the
University of Macedonia, Economic & Social Sciences

Signature of Author .....
Department of Applied Informatics, School of Information Sciences
Nov 27, 2015

Certified by .....
Ioannis Refanidis, Associate Professor of Department of Applied Informatics
University of Macedonia, Economic & Social Sciences
Thesis Supervisor

Certified by .....
Ioannis Vlahavas, Professor of Department of Informatics
Aristotle University of Thessaloniki
Thesis Committee

Certified by .....
Amedeo Cesta, Senior Research Scientist in Artificial Intelligent and Group Lead
Institute for Cognitive Science and Technology, Italian National Research Council
Thesis Committee

Planning Individual Activities through an Intelligent Calendar

by

Anastasios Alexiadis

Submitted to the Department of Applied Informatics
School of Information Sciences
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Applied Informatics

Abstract

A common use of calendars is the organization of an individual's time. While electronic calendars offer many benefits to their users, little work has been done to produce an Intelligent Calendar capable of managing the user's activities on his behalf. Towards the goal of producing the electronic calendar of the future, a calendar that will not only remind the user of his activities, but will also schedule them on his behalf, according to his preferences and the nature of the activities themselves (as well as their geographical locations), this thesis explores a model for this task. The problem explored is a complex Constraint Optimization Problem and various methods are applied on this problem (such as Constraint Logic Programming and Genetic Algorithms). An algorithm is presented, that produces the best results, based on a combination of the Squeaky Wheel Optimization Framework and a modified Simulated Annealing empowered with Tabu Search and backtracking, and coupled with in-domain heuristics, that manages to produce excellent results in a short amount of time. In addition, the problem of producing multiple plans is tackled, by defining a formal model for the quantification of the differences between plans. The scheduler presented utilizes this model to produce multiple plans for the user, while also attempting to learn his qualitative preferences from the user's plan choice. Moreover, we present SelfPlanner 2.7, a prototype of our vision for the electronic calendar of the future, which is built on top of the scheduler. SelfPlanner can be used either as a stand-alone electronic calendar application or through an Application Programming Interface (API) from other electronic calendars or applications, so as to plan individual activities through an intelligent calendar.

Thesis Supervisor: Ioannis Refanidis
Title: Associate Professor

Typeset in Hoefler Text
by Anastasios Alexiadis
using XƎLaTeX

Acknowledgments

I would like to thank my advisor, Associate Professor Ioannis Refanidis, for all his guidance during those years. I have learned much from him. I would also like to thank my family—and especially my father—for their encouragement and patience during those long years the research presented in this thesis was conducted.

Part of this work was supported 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.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Calendars can be used for the organization of one's time. They are important tools for office workers in particular, such as secretaries and office principals. Calendars are used to record time utilization, particularly the times of appointments, jobs to do and meetings. Paper Calendars are available in various forms and styles. There are desk calendars, wall calendars, even pocket calendars.

However the main problem in their successful use for organization is the task of managing the user's time, either the calendar user's or another's that the user manages on his behalf (such as in the case of secretaries), that is the planning aspect of time utilization. The calendar can be recorded after each entry has been decided. There is *planning* involved by the calendar user for the successful decision of each entry, that is the user has to decide on the date, time, duration and location of the entry. The calendar provides the means for the recording of the decided events, while being the medium where the already recorded events can be read back as a reminder for its user. The task of planning said events has to been done solely by the person using the calendar.

During the Information Age, people began moving to electronic calendars. Early implementations of electronic calendars have been included as a part of integrated electronic office systems for a long time [68]. Like their paper versions they come in a variety of styles and structures. One early example, created by a group in MIT, is PCAL [52, 51]. It is an electronic calendar with a text-only user interface. It allows the user to add an event (which it terms as "appointment"), at a specific date and time and with an optional end time and a list of participants. It also allows the user to add *notes* (which modern electronic calendars usually refer to as "TO-DOs"), which can be associated with a date but not with a specific time. The user can add or cancel events with commands such as "schedule appointment" or "cancel 1-1 noon".

The main advantages of PCAL over traditional paper calendars are the ability to look into other people's calendars without communicating with them directly and by the utilization of the advantages of a computer (such as fast searching and copying of the calendar). The user can also change the view of the calendar from day-view to week-view and backwards, which

is something that requires two different paper calendars and the user manually copying the events he has added on one of them to the other. While the above features are advantages over traditional calendars, the main problem, which is the planning aspect of time utilization, is not tackled by this early program and little progress has been made towards this goal since.

A computer can be used to automate the planning aspect of time utilization of the user's jobs to do in an electronic calendar, provided the user has provided enough information about the jobs themselves and her preferences. What is needed is an appropriate model for the problem itself and an algorithm for solving instances of it. Towards the goal of formulating the problem, let us first categorize the main events a user adds to her calendar. A user can add to her calendar events that belong to one of the following categories:

- Events that have a specific start and end time.
- Events that do not have a specific start and end time, but rather a list of possible time intervals.
- Meetings.

In the first category we have events that have to be scheduled at a specific point of time and cannot be moved to another. These are the easiest to plan, they will either be included in the calendar at their appropriate date and time or not be included at all (in case of a conflict with another event that requires the same time slot). For example, a student has to attend a lecture in her university at a specific date and time. She cannot move the lecture to another time. In the second category, we have events that do not have to be scheduled at a specific point of time. These are more flexible than the ones in the first category. This category consists of events where the user has a choice where to place them in the calendar. Some of these events may have to be split into more events before they are concluded. For example, a researcher has to write a paper. She adds a number of events to her calendar about working on her paper. These events do not have to be scheduled at the same time or have the same duration. Moreover, they can be moved to different times if a new event arises that requires their time-slots. Last of all, in the third category we have events that do require coordination with others for their scheduling. These consist of meetings and appointments.

What makes the planning process complicated is the relations that can be formed between different events, as well as the user's preferences for the way his events are scheduled. For example, the researcher in the above example may need to complete the event "visit library" before she can start working on her paper. She may want her events spread evenly on her calendar or may instead prefer periods of heavy work, which consist of many events scheduled one after the other with short breaks between, followed by periods of rest. She may prefer to work many hours at a time on her paper, but she will have to compromise some days so she can attend to her other activities. Another source of complexity is the handling of physical locations. Some *activities* (this is the term we will use to differentiate from static calendar events)

may require the person being physically present in specific locations. Traveling times between activity locations should be taken into account when generating a plan.

Automation of cognitive tasks belongs to the field of Artificial Intelligence (AI). This thesis draws on the field of AI, and more specifically on the field of Planning and Scheduling [110, 45, 106, 44] to address this problem, that is the problem of *Planning Individual Activities through an Intelligent Calendar*. We base our problem formulation on the one presented in the article *"A Constraint-Based Approach to Scheduling an Individual's Activities"* [105]. What follows is a formal formulation of the underlying computational problem.

1.1 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 $p_i \geqslant 1$ when T_i has been scheduled. T_{ij} denotes the *j-th* part of T_i, $1 \leqslant j \leqslant p_i$. Non-interruptible activities are scheduled as one part, that is, $p_i = 1$.

$$\forall T_i : p_i = \begin{cases} 0 & \text{if } T_i \text{ is not scheduled} \\ 1 & \text{if } T_i \text{ is not interruptible} \\ \geqslant 1 & \text{if } T_i \text{ is interruptible} \end{cases} \tag{C1}$$

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,

$$\forall T_i : \sum_{j=1}^{p_i} d_{ij} = d_i \tag{C2}$$

which must be at least d_i^{min}, or 0 if T_i is not scheduled.

$$\forall T_i : \; d_i^{min} \leqslant d_i \vee d_i = 0 \tag{C3}$$

For each T_i we define the minimum and maximum possible part duration, $smin_i$ and $smax_i$,

$$\forall T_{ij} : smin_i \leqslant d_{ij} \leqslant smax_i \vee d_{ij} = 0 \tag{C4}$$

as well as the minimum and maximum allowed temporal distances between every pair of parts,

$dmin_i$,

$$\forall T_{ij}, T_{ik}\ j \neq k, d_{ij} > 0, d_{ik} > 0 : t_{ij} + d_{ij} + dmin_i \leqslant t_{ik} \vee t_{ik} + d_{ik} + dmin_i \leqslant t_{ij} \quad \text{(C5)}$$

and $dmax_i$.

$$\forall T_{ij}, T_{ik}\ j \neq k, d_{ij} > 0, d_{ik} > 0 : t_{ij} + dmax_i \geqslant t_{ik} + d_{ik} \wedge t_{ik} + dmax_i \geqslant t_{ij} + d_{ij} \quad \text{(C6)}$$

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.

$$\forall T_{ij}, d_{ij} > 0 : \exists k, 1 \leqslant k \leqslant F_i, a_{ik} \leqslant t_{ij} \leqslant b_{ik} - d_{ij} \quad \text{(C7)}$$

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 $l_{ij} \in Loc_i$,

$$\forall T_{ij} : l_{ij} \in Loc_i \quad \text{(C8)}$$

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.

$$\begin{aligned} &\forall T_{ij}, T_{mn}, T_{ij} \neq T_{mn}, d_{ij} > 0, d_{mn} > 0 : (Dist(l_{ij}, l_{mn}) > 0 \vee Dist(l_{mn}, l_{ij}) > 0) \\ &\implies t_{ij} + d_{ij} + Dist(l_{ij}, l_{mn}) \leqslant t_{mn} \vee t_{mn} + d_{mn} + Dist(l_{mn}, l_{ij}) \leqslant t_{ij} \end{aligned} \quad \text{(C9)}$$

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 between them) and the sum of their utilization values should not exceed the one unit.

$$\forall t : \sum_{\substack{T_{ij} \\ t_{ij} \leqslant t < t_{ij} + d_{ij}}} utilization_i \leqslant 1 \quad \text{(C10)}$$

The model supports four types of binary constraints between activities: Ordering con-

straints, 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.

$$\forall T_i, T_j, d_i > 0, d_j > 0 : T_i < T_j \implies \forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0 : t_{ik} + d_{ik} \leqslant t_{jl} \tag{C11}$$

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.

$$\forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0, dmin_{ij} > 0 : t_{ik} + d_{ik} + dmin_{ij} \leqslant t_{jl} \vee t_{jl} + d_{jl} + dmin_{ij} \leqslant t_{ik} \tag{C12}$$

$$\forall T_{ik}, T_{jl}, d_{ik} > 0, d_{jl} > 0, dmax_{ij} < \infty : t_{ik} + dmax_{ij} \geqslant t_{jl} + d_{jl} \wedge t_{jl} + dmax_{ij} \geqslant t_{ik} + d_{ik} \tag{C13}$$

Finally, an implication constraint of the form $T_i \implies T_j$ implies that in order to include T_i in the plan, T_j should be included as well.

$$\forall T_i, T_j : T_i \implies T_j \iff (d_i > 0 \implies d_j > 0) \tag{C14}$$

Scheduling individual activities is considered a constraint optimization problem. That said, the empty schedule is a valid schedule but with low utility (an empty schedule may have some utility due to implication preferences), thus we are interested in better schedules. 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:

$$U_i(d_i) = \begin{cases} 0, & \text{if } d_i < d_i^{min} \\ u_i^{low} + \frac{(d_i - d_i^{min})}{(d_i^{max} - d_i^{min})}(u_i^{high} - u_i^{low}), & \\ & \text{if } d_i^{min} \leqslant d_i \leqslant d_i^{max} \\ u_i^{high}, & \text{if } d_i > d_i^{max} \end{cases} \tag{1.1}$$

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

Each activity T_i included in the schedule contributes utility $U_i(d_i)$ (Formula 1.1) that depends on its allocated duration. The way T_i is scheduled by a schedule π_i within its temporal domain constitutes another source of utility, $U_i^{time}(\pi_i)$. The user can define arbitrary mono-

tonic 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. 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:

$$U_i^{dmin}(\pi_i) = \frac{\sum_{t \in \pi_i} \sum_{t' \in \pi_i, abs(t-t') \geqslant pdmin_i} 1}{d_i \times d_i} \times u_i^{dmin} \tag{1.2}$$

$$U_i^{dmax}(\pi_i) = \frac{\sum_{t \in \pi_i} \sum_{t' \in \pi_i, abs(t-t') \leqslant pdmax_i} 1}{d_i \times d_i} \times u_i^{dmax} \tag{1.3}$$

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 $DoS(p)$, 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.

$$DoS(p) = \frac{\sum_{t_i \in \pi_i} \sum_{t_j \in \pi_i, p(t_i, t_j) holds} 1}{d_i \times d_j} \tag{1.4}$$

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:

$$U = \begin{array}{c} \displaystyle\sum_{i}^{d_i \geqslant d_i^{min}} (U_i(d_i) + U_i^{time}(\pi_i) + U_i^{dmin}(\pi_i) + U_i^{dmax}(\pi_i)) \\ + \\ \displaystyle\sum_{p(T_i,T_j)\in P} u_p \times DoS(p(T_i, T_j)) \end{array} \tag{1.5}$$

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.

1.2 An Example

Let us define a trivial activity scheduling problem instance as an example. Suppose an academic has the following activities he wants to define in the above model:

1. *Write an academic paper*. He can work on it during office hours in his office. He predicts he will need between 8–16 hours to complete it. He can spend as little as one hour at a sitting, or as many as three hours and wants these sessions to be spaced at least three hours (but no more than one day) away from each other. He can not start working on the paper until he has visited the library first. He wants to complete the paper until the end of the week. This is an interruptible activity.

2. *Visit the library* (between the hours the library is open). He will need to stay there between two to three hours.

3. *Give a lecture* on Monday afternoon between 15:00–17:00.

4. *Be available at his office* on Tuesday between 12:00–13:00 for student visiting hours.

5. *Have lunch* every day between 12:00–15:00. He spends 30 minutes to one hour each day on lunch. He can have his lunch either at the university restaurant or at home.

In addition he wants a plan with a lot of free time each day, with his activities spread as far away from each other as possible, while not having to spend more than one day without working on the paper. The current time is Monday morning 10:00am and the current location is the academic's office. As time is discrete in the model we will use a time resolution of 30 minutes (i.e., 0 is Monday 10:00am, 1 is Monday 10:30am, 2 is Monday 11:00am etc.).

We define 9 activities $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9\}$, with the first four corresponding to the first four user activities and the last five to the fifth user activity (one for each day between Monday to Friday). For T_1 we define $d_1^{min} = 16$, $d_1^{max} = 32$, $smin_1 = 2$, $smax_1 = 6$, $D_1 = [0, 16) \cup [48, 64) \cup [96, 112) \cup [144, 160) \cup [192, 208)$, $dmin_1 = 6$, $dmax_1 = 48$, $u_1^{low} = 100$, $u_1^{high} = 180$, $utilization_1 = 1$ and $Loc_1 = L_1$. For T_2 we define $d_2^{min} = 4$, $d_2^{max} = 6$, $smin_2 = 4$, $smax_2 = 6$, $D_2 = [0, 16) \cup [48, 64) \cup [96, 112) \cup [144, 160) \cup [192, 208)$,[1] $dmin_2 = 0$, $dmax_2 = \infty$, $u_2^{low} = 100$, $u_2^{high} = 200$, $utilization_2 = 1$, $Loc_2 = L_2$ and $p_2 = 1$.[2] For T_3 we define $d_3^{min} = d_3^{max} = 4$, $smin_3 = smax_3 = 4$, $D_3 = [10, 14)$, $dmin_3 = 0$, $dmax_3 = \infty$, $u_3^{low} = u_3^{high} = 200$, $utilization_3 = 1$, $Loc_3 = L_1$ and $p_3 = 1$. For T_4 we define $d_4^{min} = d_4^{max} = 2$, $smin_4 = smax_4 = 2$, $D_4 = [50, 52)$, $dmin_4 = 0$, $dmax_4 = \infty$, $u_4^{low} = u_4^{high} = 100$, $utilization_4 = 1$, $Loc_4 = L_1$ and $p_4 = 1$. For T_5 we define $d_5^{min} = 1$, $d_5^{max} = 2$, $smin_5 = 1$, $smax_5 = 2$, $D_5 = [4, 10)$, $dmin_5 = 0$, $dmax_5 = \infty$, $u_5^{low} = 100$, $u_5^{high} = 120$, $utilization_5 = 1$, $Loc_5 = \{L_1, L_3\}$ and $p_5 = 1$. T_6 is defined similarly but with $D_6 = [52, 58)$.[3] We define T_7–T_9 in the same way.

We have 3 locations $L = \{L_1, L_2, L_3\}$, with L_1 being the university where the academic's office is also located, L_2 being the location of the library and L_3 being the location of his house. Assuming the university is located one hour away from his house and half an hour away from the library, while his house is located half-an-hour away from the library we define the location matrix as follows:

$$Dist = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

We also define an implication and an ordering constraint $C = \{T_1 \Rightarrow T_2, T_2 < T_1\}$. To model the academic's preference for spreading the activities over the whole week we define set P with the following binary proximity preferences:

$$P = \{\forall T_{ik}, T_{jl}\ i \neq j : bpdmin_{T_{ik}, T_{jl}} = \frac{width_D}{\sum_{x=1}^{9} p_x}\}$$

which we set to 5 utility points for each $bpdmin_{T_{ik}, T_{jl}}$, as well as $pdmin_1 = \frac{width_{D_1}}{p_1}$, with 5 utility points again for T_1.[4]

[1] Assuming the library is open during office hours (Monday to Friday 10:00–18:00).

[2] Defining it as a non-interruptible activity.

[3] Corresponding to Tuesday 12:00–15:00.

[4] T_1 is an interruptible activity so we need to set a proximity preference for spreading its activity parts over the activity's domain as well.

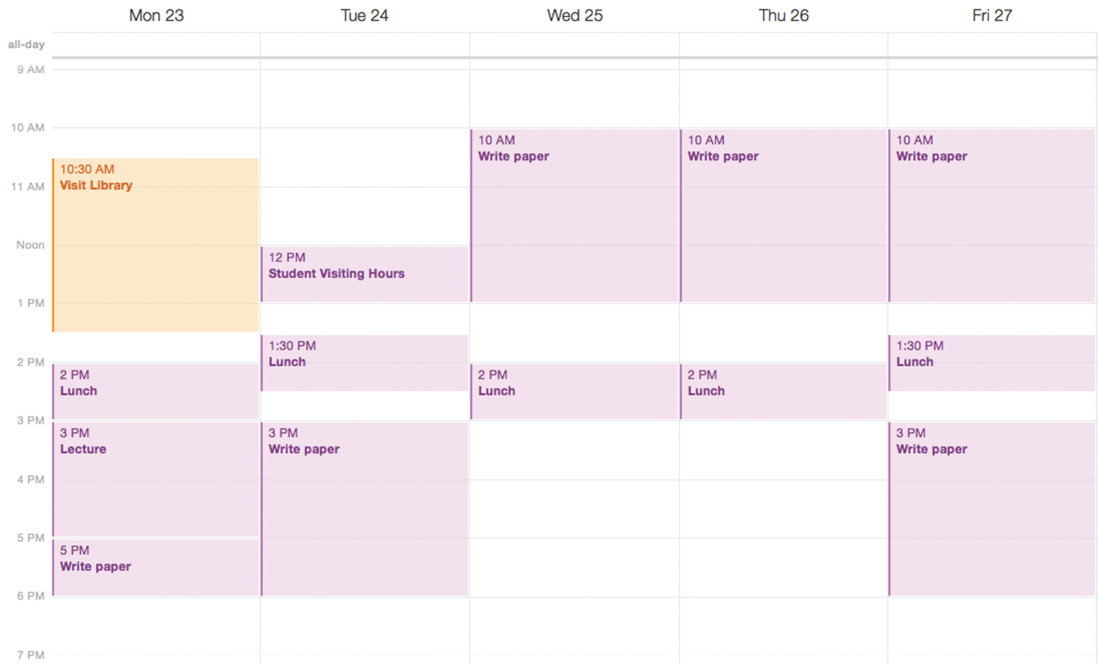

Figure 1-1: A possible plan

A possible solution for this problem instance is: $p_1 = 6, t_{11} = 14, d_{11} = 2, l_{11} = L_1, t_{12} = 58, d_{12} = 6, l_{12} = L_1, t_{13} = 96, d_{13} = 6, l_{13} = L_1, t_{14} = 144, d_{14} = 6, l_{14} = L_1, t_{15} = 192, d_{15} = 6, l_{15} = L_1, t_{16} = 202, d_{16} = 6, l_{16} = L_1, t_{21} = 1, d_{21} = 6, l_{21} = L_2, t_{31} = 10, d_{31} = 4, l_{31} = L_1, t_{41} = 50, d_{41} = 2, l_{41} = L_1, t_{51} = 8, d_{51} = 2, l_{51} = L_1, t_{61} = 55, d_{61} = 2, l_{61} = L_1, t_{71} = 104, d_{71} = 2, l_{71} = L_1, t_{81} = 152, d_{81} = 2, l_{81} = L_1, t_{91} = 199, d_{91} = 2, l_{91} = L_1, p_2 = p_3 = \ldots = p_9 = 1$. It is illustrated in Figure 1-1.

1.3 Contributions of This Thesis

The contributions of this thesis concern planning individual activities through an intelligent calendar. Towards the goal of achieving the electronic calendar of the future, a calendar where the user will not only record her events and view them back, but have the calendar itself plan her activities and produce her schedule—taking into account all her preferences and the traveling time between the locations of her activities—we consider the following aspects of such a system: We consider the formal model for defining an individual activity scheduling problem, and extend it with support for non-monotonic temporal preferences. This extension enriches the model, by allowing the user to define more complex temporal preferences. For example, a user can define arbitrary non-monotonic functions of time over the temporal domain that give higher utility when an activity's parts are scheduled during specific time periods (such as mornings, near noon, specific days etc.). In addition, we extend the model so as to minimize the total time spend traveling between locations by the user.

We consider different algorithms for solving the Constraint Optimization Problem (COP), defined by the above model, and evaluate them. One of them is a complete solver that manages to produce the *exact* global maximum solution for a problem instance, that we implemented in ECL^{i}PS^{e} Prolog [7]. We also consider non-complete and nondeterministic algorithms, such as genetic algorithms for solving the problem. We decide upon a combination of an implementation of the Squeaky Wheel Optimization Framework (SWO) and a modified Simulated Annealing, a local search stochastic function minimizer that is motivated by the physical process of annealing, empowered with Tabu Lists and backtracking and coupled with in-domain heuristics, that produces excellent solutions in a relative short amount of time. This algorithm, which we present in Chapter 3 (Section 3.4 onwards), along with a thorough experimental evaluation, will appear in an article in the *"AI Communications"* journal which is entitled *"Optimizing Individual Activity Personal Plans through Local Search."*

We also consider the extension of the scheduler, produced by the above work, to produce multiple solutions for a given problem instance. To successfully do this we formally define functions that quantify the difference of plans produced for a given problem instance, and implement them to extend the scheduler to produce multiple plans for a given problem instance. As the quantification of a user's preferences for her plans is a hard problem, we also consider a machine learning algorithm that allows the user to evaluate her plans qualitatively and have the scheduler attempt to learn her preferences online (i.e., the more she uses the scheduler, the more the scheduler will produce plans that converge with the user's actual preferences). This research, which is described in Chapter 4, provides an article under review entitled *"Alternative Plan Generation and Online Preference Learning in Scheduling Individual Activities."*

We also present SelfPlanner 2.7, a prototype electronic calendar application that utilizes the research produced in this thesis, to produce a prototype for our vision of the electronic calendar of the future. SelfPlanner, which can be used either as a stand-alone electronic calendar application,[5] or through an Application Programming Interface (API) for use by other electronic calendar applications and programs. It greatly simplifies the problem model (as presented in Section 1.1), and provides a higher-level view of it to the user (both the end-user using its client[6] and the programmer using its API). myVisitPlanner [103], a system founded by a research grant, uses SelfPlanner, through its API, as the system's planning engine. SelfPlanner was presented in the *International Conference of Planning and Scheduling (ICAPS) 2013 application showcase program*. The system's original deployed version, along with its evaluation, which is described in Chapter 5, was published in an article in the *"Computational Intelligence"* journal which is entitled *"Deployment and Evaluation of SelfPlanner, an Automated Individual Task Management System."*

SelfPlanner started as the author's master thesis and was later expanded and evaluated as part of this thesis. A minor contribution is the algorithm that SelfPlanner uses for its

[5] It is not completely standalone. It utilizes Google Calendar to present its plans, providing thus all the benefits of this modern electronic calendar, and Google Maps to specify real-world locations.

[6] SelfPlanner is based on a client-server model.

templates, presented in Section 6.5, and published in the *"Proceedings of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI)"*. The paper is entitled *"Defining a Task's Temporal Domain for Intelligent Calendar Applications."*

Finally, we consider extending SelfPlanner with joint activity scheduling, that is producing plans that include activities which involve more than one user, and present future directions for this research.

1.4 Structure of This Thesis

The rest of this thesis is structured as follows: In Chapter 2 we discuss the literature. First, we discuss the academic field of Planning and Scheduling. We proceed to discuss Constraint Satisfaction Problems (CSP) and Mathematical Optimization. Subsequently we discuss Constraint Logic Programming. We then proceed to discuss non-exact local search methods and metaheuristic algorithms. Finally we discuss the Squeaky Wheel Optimization Framework.

In Chapter 3 we explore different algorithms for solving the Constraint Optimization Problem discussed in Section 1.1. Particularly we explore Constraint Logic Programming and Genetic Algorithms to produce four schedulers for the problem.[7] Moreover, we extend the problem formulation by defining non-monotonic temporal preferences. Based on our insights from the previous sections, we discuss the algorithm we propose for solving the problem and compare it thoroughly with SWO on the original problem instances where SWO was benchmarked [105]. We also discuss our findings.

In Chapter 4, we discuss our extensions to the scheduler of the previous chapter for calculating multiple plans. We also introduce an on-line machine learning method that attempts to learn the user's preferences. We then proceed to evaluate our extensions.

In Chapter 5, we present the original deployed version of the SelfPlanner electronic calendar prototype along with its evaluation.

In Chapter 6, we describe the current version of SelfPlanner, that is SelfPlanner 2.7. We also briefly discuss the myVisitPlanner system. In addition, we present the Object-Oriented model of SelfPlanner's API, which is used by both the SelfPlanner client and by myVisitPlanner. Moreover, we present another extension to the problem formulation that minimizes the total traveling time of a user in a plan. Last of all, we discuss joint activity scheduling and present a protocol for the negotiation of different parties participating in a joint activity.

Finally, in Chapter 7 we conclude this thesis and provide future directions for research on this problem.

[7] One of them solves a subset of the problem.

CHAPTER 2

Background

More than fifty years have passed since the General Problem Solver (GPS) [91], the first planning system, was designed. GPS was an ambitious project, with a goal of solving *all* formalized symbolic problems. Although it did not live up to expectations, it was still an important contribution for a variety of reasons, the most important of which was that it was the first program that separated its *knowledge* of particular problems from its problem-solving *strategy* [95] (i.e., the idea to use a generic solver). In 1971, Fikes and Nilsson presented the STRIPS planner [40], probably the single most important contribution to the field of planning. The STRIPS planner presented a problem modeling language, with the same name, based on an arbitrary collection of first-order predicate calculus formulas, that is still in use today.

From the 1960s, the problem of planning actions was considered as a search problem on the set of all possible states of a problem space [120].[1] As the problem space is often too complicated to perform an exhaustive search, one needs to define appropriate search strategies (i.e., heuristics),[2] to guide the search to an appropriate solution (i.e., one of the goal states).

The rest of this chapter is structured as follows: First we explore the field of Planning. Then we proceed to explore Scheduling and Constraint Satisfaction Problems. Subsequently, we discuss Mathematical Optimization. We proceed to discuss Constraint Logic Programming and Branch and Bound algorithms. After which, we discuss local search methods and metaheuristic algorithms. Finally we discuss the SWO algorithm.

2.1 Planning

Russel and Norvig give the following definition of planning [110]:

[1] Simon and Newell separated the notion of the *task environment* from the *problem space*, the first being the problem space as viewed by an omniscient observer, while the later being a limited view of the problem by a particular subject.

[2] A heuristic function is the most common form in which additional knowledge of the problem is inserted into the search algorithm [110]. A heuristic is said to be *admissible* if it never overestimates the cost of reaching the goal.

The task of coming up with a sequence of actions that will achieve a goal is called *planning*.

Fundamentally, planning is a synthesis task [122]. A sequence of a actions (out of a set of possible actions for a particular problem) must be selected in a particular order. When this sequence is applied on an *initial* state, it must lead to one of the *goal* states. A problem instance is defined by defining these initial and goal states. The actions themselves have logical interactions between them.

Planners offer the following advantages to traditional search algorithms when working on problems that can be formulated in their model: They include heuristics that operate on the planning model itself, without knowledge of the particular problem being applied to. The next advantage is problem decomposition. A planner can decompose the problem of reaching a goal state, into smaller planning problems involving reaching various subgoals. Problems having a worst case complexity of $O(n!)$ can be decomposed into problems having a worst case complexity of $O((n/k)! \times k)$ [110]. These sub-plans can later be compiled into a plan that reaches the goal state. Even for problems that are nearly decomposable there exist planners (i.e., partial-order planners) that can still perform problem decomposition.

Smith et al (2000) [122] categorize most of the planning work in AI into five camps. These are: Hierarchical Task Network (HTN) planning, and Decision-theoretic Planning, case-based planning and reactive planning, with the last two being less relevant for combinatorial optimization problems (i.e., finding an optimal object from a finite set of objects [114]). These approaches have been explored in great detail [130, 131, 35, 17, 19].

The two most common modeling languages for planning problems are STRIPS [40] and the Action Description Language (ADL) [99].[3] States are represented as a conjunction of literals.[4] For example, $Careful \wedge Observant$ may represent the status of an exploring agent. The initial state is a fully specified state, while the goal is represented as a partially specified state. An action is specified in terms of preconditions that must be satisfiable for the action to be applicable on a state, and in terms of effects that alter the state when the action has completed executing. For example one could specify the action "travel by train" in the STRIPS language as:

$$
\begin{aligned}
&Action(Travel\text{-}by\text{-}train(p, from, to),\\
&\quad \textsc{precond}{:}At(p, from) \wedge Train(p) \wedge Train\text{-}station(from) \wedge Train\text{-}Station(to)\\
&\quad \textsc{effect}{:}\neg At(p, from) \wedge At(p, to)
\end{aligned}
$$

ADL extends STRIPS by providing a more expressive modeling language with support for quantified variables in goals, condition effects, an open world assumption, allowing goals to be

[3] The current standard syntax is called the Planning Domain Definition Language (PDDL) [78], which includes sublanguages for STRIPS, ADL and for HTNs [110].

[4] STRIPS is limited to positive literals.

represented with both conjunctions and disjunctions etc. [110].

There are numerous methods to solve planning problems, starting with simpler ones like forward state space search to more advanced ones like GRAPHPLAN [15]. Older algorithms searched the space of possible *states*, while newer ones operate on the space of possible *plans*. GRAPHPLAN operates on two phases, the *graph expansion* phase and the *solution extraction phase*. The planning graph consists of two types of nodes, the proposition or fact nodes and the action nodes, arranged in alternative levels. At each level of proposition nodes there exist all propositions that can hold because there are actions in the previous level that produce them as their effects. At each level of actions there exist all actions that can be applied on some subset of propositions from the previous level. In the solution extraction phase the problem is transformed into a Propositional Satisfiability Problem (SAT, see Section 2.3.1) that can be solved by a SAT solver.

Hierarchical Transition Network (HTN) planning is used by most planning systems that have been developed for practical applications [132, 126, 88]. In HTN planning high-level tasks can be defined which consist of primitive tasks. For example one could define the high-level task `WriteBook(?bibliography, ?diagrams)` which consists of `TakeNotes(?bibliography)`, `Typeset(?diagrams)` and `TypeBook(?bibliography, ?diagrams)`. A high-level task is described by a transformation rule called a *method*, a mapping from the high-level task into a partially ordered network of low-level tasks together with a set of constraints. The planning process recursively expands high-level tasks into networks of lower-level tasks [122].

2.2 Scheduling

Scheduling is the problem of assigning limited resources to tasks over time so as to optimize one or more objectives [8, 100]. The common concept in AI is that scheduling problems are planning problems where the actions are predefined, leaving only the problem of determining a feasible order, which Smith et al (2000) [122] consider a trivialization of the problem. The main difference between scheduling in Operations Research (OR) and scheduling in AI, is that the researchers of the first reason about and analyze specific domains of scheduling problems so as to produce better models of these domains, while the researchers of the later tend to explore general representations and methods that cover a range of different types of scheduling problems.

The most common approach in AI has been to represent scheduling problems as Constraint Satisfaction Problems (CSP) and solve them using general constraint satisfaction methods [10, 32, 123]. This approach misses the fact that finding a *good* schedule is much harder than finding a valid schedule. Moreover scheduling problems also involve choices, such as what resource to use for a task or which task to place in a limited temporal domain—choices that may have different costs and durations.

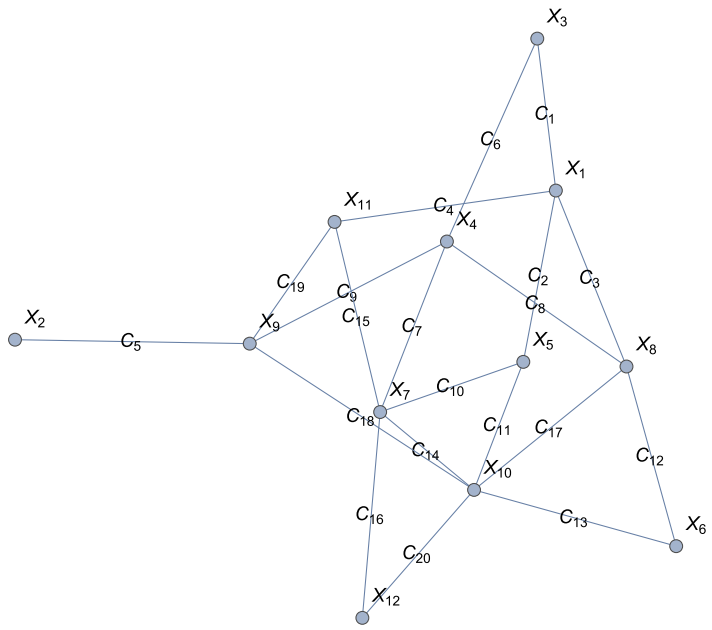

Figure 2-1: A CSP network with 12 variables and 20 binary constraints. The most constrained variable is X_{10} and the least constrained is X_2

2.3 Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) [83, 110] is a satisfiability problem that can be defined as a triple $\langle X, D, C \rangle$ where:

$$
\begin{aligned}
X &= \{X_1, \ldots, X_n\} && \text{is a set of decision variables} \\
D &= \{D_1, \ldots, D_n\} && \text{is a set of the domain of values of the variables} \\
C &= \{C_1, \ldots, C_m\} && \text{is a set of constraints}
\end{aligned}
$$

A solution of a CSP is for every variable X_i ($1 \leqslant i \leqslant n$) to be assigned one value from its domain D_i, while satisfying all the constraints in C. In this definition we will only consider unary and binary constraints, that is constraints between two variables ($C_k = c(X_i, X_j)$). It is possible to convert any CSP with n-ary constraints into an equivalent CSP with unary and binary constraints [72, 109]. It is also possible to define high-level constraints, which are termed *global constraints*. A global constraint is a constraint that encapsulates a set of other constraints. Auxiliary variables can also be used to define more complex constraints.

A CSP network is the graph formed by considering the decision variables as the nodes of the graph and the constraints as the edges connecting them (Figure 2-1). In such a network the following levels of consistency are defined:

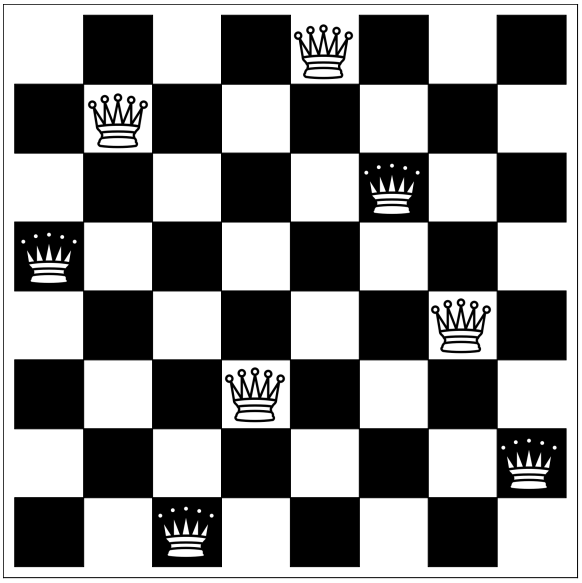

Figure 2-2: A solution for the 8 queens problem

Node consistency: $\forall v \in D_i : c_i(v)$

Arc consistency: $\forall v \in D_i \; \exists w \in D_j : c_{ij}(v, w)$

Path consistency: $\forall v \in D_i, w \in D_j \; \exists u \in D_k : c_{ik}(v, u), c_{kj}(u, w)$

Node consistency refers to unary constraints being satisfiable, while arc consistency refers to binary constraints being satisfiable. Path consistency refers to whole paths of constraints being satisfiable. To solve CSP problems, we combine search methods using heuristics (such as picking the most constrained variable, the least constrained etc.),[5] to partition the search space into smaller sub-spaces, with constraint propagation to exclude invalid solutions from the search space [119]. To do constraint propagation we have to decide what level of consistency we want to enforce (i.e., node consistency and arc consistency).

As an example, consider the popular 8-queens problem.[6] One possible model of the problem as a CSP is the following: Let $X = \{X_1, X_2 \ldots X_8\}$, where each variable represents a column. Each variable X_i ($1 \leqslant i \leqslant 8$) has a domain $D_i = [1..8]$, representing the row where a queen will be placed. With constraints:

[5] Another important choice is the value ordering by some value choice heuristic. For example, when we pick a variable we can start assigning values from the beginning of its domain or we can start from the minimum/maximum value, the middle value, the median or start the enumeration by splitting the domain successively in halves until a ground value is reached [7].

[6] In the 8-queens problem, eight queens have to be placed on a chessboard, one at each column, with neither of the queens threatening any of the others either horizontally, vertically or diagonally. The 8-queens problem has 92 solutions.

$$
\begin{aligned}
C_1 &= \texttt{alldifferent}(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8)^7 \\
C_2 &= X_1 \neq X_2 + 1 \\
C_3 &= X_2 - 1 \neq X_1 \\
C_4 &= X_1 \neq X_3 + 2 \\
&\vdots \\
C_{57} &= X_8 - 7 \neq X1
\end{aligned}
$$

C_1 does not allow the queens to be placed at the same row, while constraints C_2–C_{57} do not allow each queen from having another queen in the same diagonal. A solution to the problem is illustrated in Figure 2-2.

2.3.1 Propositional Satisfiability Problems

Propositional Satisfiability Problems (SAT) are satisfiability problems where the variables are defined over the boolean domain. The constraints are defined using propositional logic. It is also possible to model linear constraints in this form [37].

A SAT problem is defined in Conjunctive Normal Form (CNF), that is as a conjunction of clauses. Each clause is a disjunction of literals. A SAT problem is satisfiable if there is an assignment of the variables where the formula in CNF evaluates to `true` and a solution is one of these assignments in a satisfiable problem. SAT has been proven to be a NP-complete problem [31].[8]

There are various methods to solve a SAT problem such as the Davis-Putnam-Logemann-Loveland procedure (DPLL), which is a complete search for finding a valid assignment of the variables or proving that is unsatisfiable. DPLL repeatedly selects an unassigned literal in the CNF and then recursively searches for a valid assignment. This procedure has been enhanced with various methods, such as clause learning (caching of clauses of conflict), variable and value heuristics, the watched literals scheme, conflicted-directed backjumping (backtracking to the level of the troublemaking variables), randomized restarts and many others. There are incomplete methods as well, such as Walksat. For a thorough review on modern SAT solvers see Gomes et al (2008) [50].

To model the 8-queens problem as a SAT, we need 64 variables, each representing a square in the chessboard, with `true` implying a queen on that square and `false` that the square is empty. One way to model the problem in CNF, would be to define clauses that require at least one of the variables of each row to be true (8 clauses), and clauses forbidding any binary combination of variables, that are either in the same row, column or diagonal, of being both true.

[7] `alldiferent` is a global constraint.

[8] If we restrict all the clauses to two or fewer literals we form the 2-SATISFIABILITY (or 2-SAT) problem. This problem has a polynomial time solution [75].

$$
\begin{aligned}
&(Q(1,1) \vee Q(1,2) \vee Q(1,3) \vee Q(1,4) \vee Q(1,5) \vee Q(1,6) \vee Q(1,7) \vee Q(1,8)) \wedge \\
&(Q(2,1) \vee Q(2,2) \vee Q(2,3) \vee Q(2,4) \vee Q(2,5) \vee Q(2,6) \vee Q(2,7) \vee Q(2,8)) \wedge \\
&\qquad \vdots \\
&(\neg Q(4,1) \vee \neg Q(1,1)) \wedge \\
&(\neg Q(4,1) \vee \neg Q(1,4)) \wedge \\
&(\neg Q(4,1) \vee \neg Q(2,1)) \wedge \\
&\qquad \vdots \\
&(\neg Q(8,8) \vee \neg Q(8,7))
\end{aligned}
$$

where $Q(i,j)$ ($1 \leqslant i \leqslant 8$, $1 \leqslant j \leqslant 8$) represents the i,jth square. The CNF consists of 1464 clauses.

2.4 Mathematical Optimization

Mathematical Optimization (also called Mathematical Programming) [134, 133] grew out the need to optimize resource distribution under physical constraints, so as to plan expenditures and returns in order to reduce costs for the army in World War II. It is primarily used in Operations Research to apply advanced analytical methods for the goal of making better decisions. Mathematical Programming (MP) is used in many other fields as well, such as economics, engineering and in the industry (e.g., telecommunications and manufacturing). It was originally termed Linear Programming but later expanded to Mathematical Programming to include the study of nonlinear models as well. In this section we will focus on linear models.

Linear Programming (LP) [21, 14, 113] can be used to solve a variety of real world problems. Its strength lies in finding the optimal solution of out of the set of feasible solutions, provided the solution evaluation function is modeled correctly in the LP's *objective function*.

In LPs a linear objective function is either minimized or maximized subject to a set of linear equalities and inequalities (constraints).[9] An LP problem can be solvable, infeasible or unbounded (this is the case where there is no limit in optimizing the objective function without violating the constraints). Consider the following maximization problem involving two variables x_1 and x_2:

$$
\begin{aligned}
\text{Maximize} \quad & x_1 + 1.5x_2 \\
\text{subject to} \quad & x_1 + 2x_2 \leqslant 4 \\
& 3x_1 + 4x_2 \leqslant 10 \\
& x_1 \geqslant 0, x_2 \geqslant 0
\end{aligned}
$$

[9] In Nonlinear Programming some of the constraints or the objective function are nonlinear.

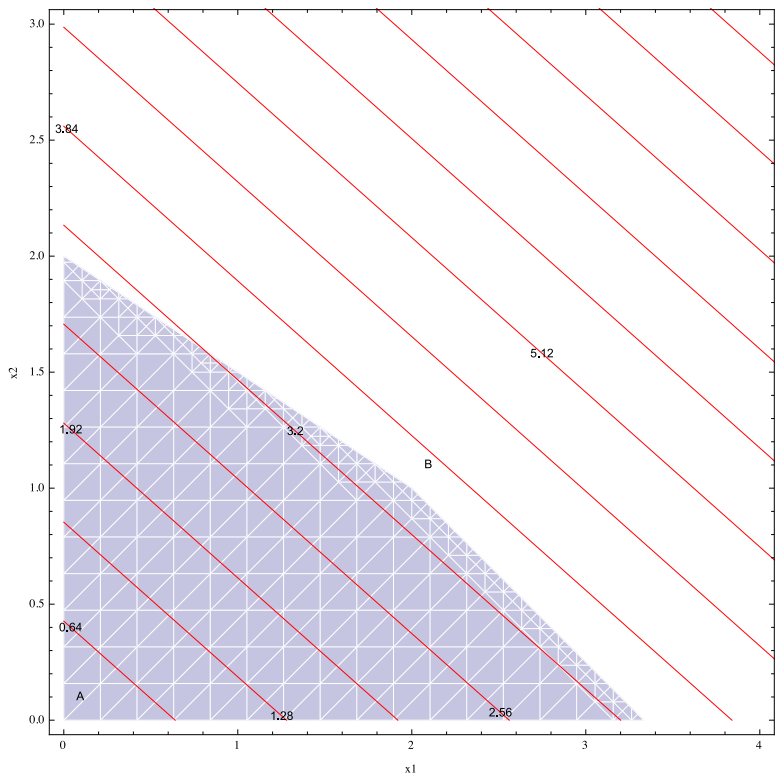

Figure 2-3: The feasible region of a Linear Programming problem

The feasible region of this problem is illustrated in Figure 2-3. The contour lines stand for the objective function at various assignments of the decision variables. This problem has an optimal solution at $x_1 = 2$, $x_2 = 1$.

LPs are usually solved either by some version of the Simplex method (published by George B. Dantzig in 1947) or by an Interior Point method algorithm. Simplex [97] is an algorithm for general LPs. The method starts by constructing an initial feasible basic solution (vertex) and converges monotonically to the optimal solution (vertex) by performing pivots. Geometrically, a pivot is the movement to an adjacent vertex. In practice, Simplex tends to perform a linear number of iterations to the number of constraints of the problem, and converge in polynomial time. However it has an exponential worst-case complexity [71, 97], while Interior Point methods have a polynomial worst-case complexity. In contrast to constraint propagation techniques, Simplex and Interior Point algorithms do not compute a representation of the reduced search space [119].

In Linear Programming the domain of the variables is defined over $\mathbb{R}$. In Integer Linear Programming (ILP) [90] the domain of the variables is defined over $\mathbb{Z}$. It is also possible to combine integer and real variables in Mixed Integer Linear Programming (MILP). ILP problems are computationally expensive to solve as they are NP-hard problems [38]. Branch and Bound is considered the most straightforward method to solve such problems [26],[10] so these problems

[10] ILPs are also solved by the cutting-plane method, branch and cut, even heuristic methods that give non-exact

are usually solved by a combination of branch and bound search with an underlying LP solver. Techniques of linear relaxation have been devised (and are utilized by ILP/MILP solvers), such as replacing the integer constraints on the variables in a 0-1 ILP by weaker constraints, that restrict the variables to the $[0..1]$ interval. This relaxation transforms this problem from NP-hard to polynomial time complexity. It is possible to model logical constraints as linear constraints on 0-1 ILPs [53, 23], as well as produce linear relaxations for CSP's global constraints [59].

To model the 8-queens problem as a Linear Programming problem, we can use an ILP with 64 variables, one for each square. A value of 1 implies a queen on that square and a value of 0 the absence of one. We can formulate the ILP as follows:

$$
\begin{array}{ll}
\text{Max} & x_1 + x_2 + x_3 + \ldots + x_{64} \\
\text{s.t.} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + 0x_9 + \ldots + 0x_{64} \leqslant 1 \\
& \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& x_1 + \ldots + x_9 + \ldots + x_{17} + \ldots + x_{25} + \ldots + x_{33} + \ldots + x_{41} + \ldots + x_{49} + \ldots + x_{57} \ldots \leqslant 1 \\
& \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& 0x_1 + \ldots + x_5 + \ldots + x_{12} + \ldots + x_{19} + \ldots + x_{26} + \ldots + x_{33} + \ldots + 0x_{64} \leqslant 1 \\
& \vdots \qquad\qquad\qquad \vdots \\
& x_1 \geqslant 0, x_2 \geqslant 0, x_3 \geqslant 0, \ldots, x_{64} \geqslant 0
\end{array}
$$

where most of the coefficients are 0. The problem is modeled with 46 linear constraints. The objective function ensures that all queens that can be placed on the chessboard will be (i.e., all eight queens). The first 8 linear constraints do not allow more than one queen to be placed in each row, while the next 8 linear constraints do not allow more than one queen to be placed in each column. The final 30 constraints do not allow queens in the same diagonal.

2.5 Constraint Programming

> "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."
>
> Eugene C. Freuder, Constraints, April 1997

Constraint Programming (CP) [6, 67, 56] is the paradigm where relations between variables are stated in the form of constraints. CP has been successfully applied in a number of fields including molecular biology, engineering and numerical analysis. It was originally defined as an extension to logic programming called Constraint Logic Programming (CLP)—which is still

results such as hill-climbing, simulated annealing, tabu search, ant-colony optimization etc.

termed when provided as an extension to a logic programming language. The concept of constraint satisfaction was formulated by AI researches in the seventies, along with the main notions of local consistency and the algorithms to solve such problems. It draws out of the field of combinatorial optimization.

CP is a more flexible modeling language than Mathematical Programming. A problem can be represented by high-level built-in and user-defined constraints. These constraints can be linear/nonlinear, logical and high-level. For example, consider the global constraint sequence_total, which restricts a set of variables to take values between a specified range for all sequences of K variables in the set, while specifying the total occurrence of each value in another range. The modeling represents a top-down approach since the constraints are defined with the use of other constraints [108]. This property of CPs allows an additional insight into the structure of the problem [34].

Every CP solver is based on constraint propagation [6], which is the enforcement of local consistency to subsets of variables or constraints. Local consistency is enforced by the application of transformations to the current subproblem in order to reduce the domain of the variables, strengthen constraints, and infer new applicable constraints. The result of this procedure is a reduction of the search space. The most common local consistency conditions were mentioned in Section 2.3.[11] In CPs many local consistency algorithms can be used to perform constraint propagation, which could be considered as an efficient *local* constraint propagation [129]. In contrast, in ILPs/MILPs a Simplex algorithm used to solve the continuous relaxation of the problem could be considered an efficient *global* constraint propagation [133]. There is an undertaking to combine the two [80], and bring the benefits of LP relaxation to CP [1, 79, 108, 26], as well as bring logic-based methods to Mathematical Programming [58].

The ECL^{i}PS^e Constraint Logic Programming System [7, 111, 93] is a superset of Prolog [124] adding support for CP at the core language. Due to Prolog's declarative language it can be used both as a modeling language for CPs, as well as a general purpose programming language. Another advantage of Prolog is that it is a homoiconic language [124],[12] as code is represented in the language's fundamental data type blurring the boundaries between code and data (i.e., it is possible to write clauses that add other clauses or remove them during runtime while using the same syntax as a clause would operate on a fact). In ECL^{i}PS^e constrained variables, supporting domains over reals and integers, are data types of the language making the modeling of CSPs intuitive. Hooker argues that the core scientific mission of Operations Research and CP is modeling [60], which is the focus in ECL^{i}PS^e. In addition to modeling and solving CSP problems, ECL^{i}PS^e supports the modeling and solving of Constraint Optimization Problems (COP). These are CSPs with the addition of an objective function that has to be minimized or maximized. These problems are solved by applying a branch and bound method on top of the normal search/constraint propagation algorithm, i.e., incrementally search for solutions better than the previous one. Moreover ECL^{i}PS^e provides interfaces [119] to external MP solvers

[11] These being node, arc and path consistency.

[12] Other examples of homoiconic languages include Lisp [135] and Wolfram Mathematica [136].

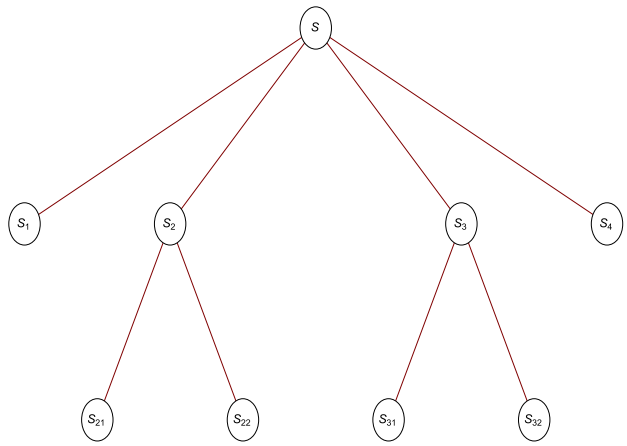

Figure 2-4: An example of the search space of B&B. S_1 and S_4 are pruned from the search in the bounding phase.

such as CPLEX, allowing one to model and solve LP/MILP problems in ECL^{i}PS^e.[13]

The constraint solvers of ECL^{i}PS^e are implemented using *suspend* goals (i.e., the simplest implementation of any constraint is to suspend until all its variables are initialized and then evaluate it). The search starts from the goal (backward chaining) and recursively scans the clauses while trying to prove the goal. Every constraint encountered in the search is placed on a set call the constraint store. The search backtracks if the set is found to be unsatisfiable [7].

ECL^{i}PS^e has its roots at systems developed during the 1980s at the European Computer-Industry Research Centre (ECRC) and was subsequently developed there until 1995 where the main development activity moved to IC-Parc at Imperial College London. In 2003 it was purchased by Cisco Systems until it was open-sourced in 2006.

2.5.1 Branch and Bound

Branch and bound (B&B) [73] is a method for solving ILPs and COPs. The method was defined in the context of combinatorial optimization [6]. Let $f(x)$ be an objective function to be either minimized or maximized and let S be the set of valid solutions (i.e., the feasible region of an ILP). The algorithm operates on two phases, the *branching* phase and the *bounding* phase. In the first phase S is split to two or more smaller sets $S_1, \ldots, S_N$, where $S = S_1 \cup \ldots \cup S_N$. A search tree is defined where S is the root node, and the subsets are the branching nodes of the tree. The minimum/maximum of $f(x)$ over S is $min/max\{v_1, \ldots, v_N\}$, where each v_i is the minimum/maximum of $f(x)$ within S_i. In the second phase, a bounding procedure computes the lower and upper bounds for the min/max value of $f(x)$ within a given subset of S. The method

[13] The limitations of LP/MILP models are in effect. These problems have to be modeled with linear constraints and all the expressive CP constraints do not apply here.

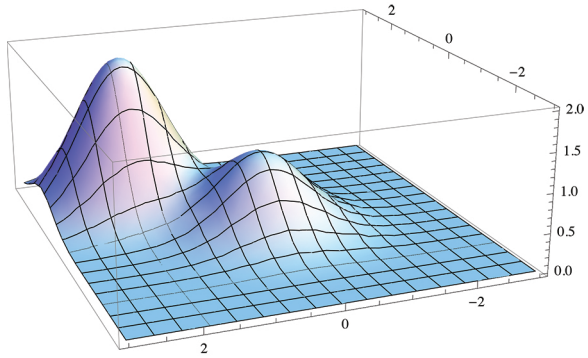

Figure 2-5: A search space with a local and a global optimum in a maximization problem.

works as follows: If the *lower* bound of a node is greater than the *upper* bound of another, in a minimization problem, or the *upper* bound of the first is lesser than the *lower* bound of another in a maximization problem, then the search tree is pruned by discarding the first node from the search. The method maintains a record of the minimum upper bound found so far, in a minimization problem, or the maximum lower bound, in a maximization problem. Any node that does not fall in that region (i.e., has a lower bound greater than the minimum upper bound in a minimization problem) is discarded from the search. The algorithm terminates when the current candidate set S is reduced to a single element or when the lower and upper bounds of S match. This method defines a *complete* search algorithm.

2.6 Local Search

Local search is a method of exploring the search space (the set of all candidate solutions) by moving via the immediate neighbors of the current solution. It is called direct search when used in continuous optimization [42, 89] where it is an alternative to gradient search methods, which use first and second order derivatives of the objective function. In contrast to gradient search methods, direct search tends to converge more slowly but is more tolerant to the presence of noise in the model or data.

To apply a local search method the neighborhood of a solution has to be defined, i.e., given a solution what its neighbor solutions are. In the most basic form a neighbor solution s' to a solution s, is s with one decision variable having a different value in its domain. The set of all possible neighbors of s define its local neighborhood. For example, consider an unconstrained optimization problem with two integer decision variables x_1 and x_2, both having a domain of $[0..1]$, and an objective function $f(x_1, x_2) = x_1 + x_2$ that has to be maximized. Let the current solution be $\{x_1 = 0, x_2 = 0\}$. Its local neighbors are $\{x_1 = 1, x_2 = 0\}$ and $\{x_1 = 0, x_2 = 1\}$, but not the optimal solution. To reach the optimal solution a local-search optimization algorithm will have to choose one of the above neighboring solutions and then move to the optimal $\{x_1 = 1, x_2 = 1\}$, which is a local neighbor of both the above solutions.

The most common local-search algorithm is *hill-climbing*, which from a given solution s, chooses the neighbor s' of s that minimizes/maximizes the objective function to move to, and recursively repeats this process until it reaches a solution where all its local neighbors have less optimal values of the objective function. Hill-climbing is a local optimizer, that is, it will terminate at the nearest local optimum. For example, if the initial solution is near the local optimum in Figure 2-5 it will reach that local optimum and terminate while there is a global optimum in the search space (or better local optimums).

2.7 Metaheuristic Optimization

Glover and Kochenberger [47] define metaheuristics as:

> Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes.

Some types of complex optimization problems may require a great length of time, in some cases even of astronomical scale, to be solved by exact methods. Metaheuristic algorithms [16, 47] succeed where many of these methods, such as standard ILP methods, fail because they are able to use strategies of greater flexibility than the ones used by methods that assure that convergence will inevitably be reached. These types of algorithms are incomplete, because they do not explore the whole search space, and many of them are nondeterministic as well, producing different solutions for the same problem instance at each run. They are able to cope with highly nonlinear and multimodal problems with different and conflicting objectives.

Metaheuristic algorithms (and optimization algorithms in general) can be classified in *trajectory* based algorithms and *population* based. Trajectory based algorithms use a single agent or a single solution at a time and trace out a path as the algorithm iterates. Some examples of trajectory based algorithms include Tabu Search and Simulated Annealing. On the other hand, population based algorithms use multiple agents or solutions and trace out many paths at each iteration. Examples of population based algorithms include Ant Colony Optimization, Particle Swarm Optimization and Genetic Algorithms. In this section we will discuss three popular metaheuristic algorithms.

2.7.1 Tabu Search

Tabu Search [46, 48] is based on the idea of forbidding (or penalizing) certain moves in the search space. The purpose of forbidding moves is to prevent cycling (i.e., prevent the search from returning to a previously evaluated solution). In this manner Tabu Search can make a move to a solution that is not the highest-utility neighbor if a higher-utility solution was previously explored (i.e., is *tabu*). The moves that are held tabu are a small fraction of the search space and a move will loose its tabu status and become accessible after an amount of time.

Tabu Search uses a *Tabu List*, to keep a record of recently visited solutions in the search space. When considering the move to a new neighbor solution, Tabu Search first checks if that solution is included in the Tabu List and if it is included it does not make the move. The oldest solution recorded in the Tabu List will be removed as soon as the Tabu List reaches a certain size and a new solution is added to the list. There are also extended versions of Tabu Search, that record intensification rules that drive the search towards more promising areas, and record diversification rules to push the search into new regions of the search space.

2.7.2 Genetic Algorithms

Genetic Algorithms [49] and Differential Evolution algorithms [125] are stochastic population based methods that attempt to mimic the physical process of evolution. They are related methods but with some important distinctions between them [55].

In Genetic Algorithms a set of solutions (termed *chromosomes*) is sampled over the search space. This set defines the *initial population*. These solutions are evaluated with the use of a *fitness* function (i.e., the objective function). At each iteration pairs of solutions are selected and from each pair two new solutions are produced using a *crossover* procedure. Some solutions are altered stochastically using a *mutation* procedure. The population set is updated to the new solutions (maybe with some of the old solutions included as well) at the end of the iteration and have their fitness values measured. This procedure is repeated until a termination condition is satisfied. The crossover and mutation procedures depend on the *encoding scheme* selected for the particular problem.

2.7.3 Simulated Annealing

Simulated Annealing [70, 61] is a stochastic trajectory based method that simulates the physical process of annealing. In annealing metals and glass are hardened by first heating them to high temperatures and then cooling them gradually so as to stabilize in a crystalline state of low energy.

At each iteration the algorithm selects a neighbor with probability:

$$e^{-\frac{\Delta E}{T_k}}$$

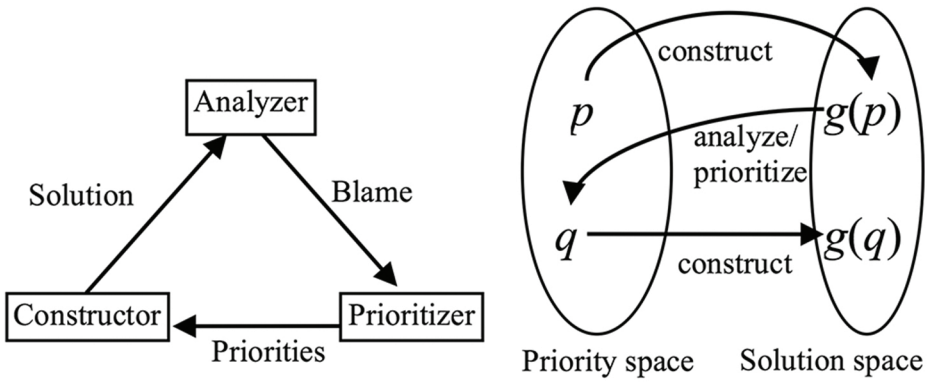

Figure 2-6: (a) The SWO cycle. (b) Coupled search spaces

where T_k is the current temperature (at the k-th iteration), which is defined by a *cooling schedule*, and ΔE is the change of energy between the neighbor and the current solution defined by:

$$\Delta E = \gamma \Delta f$$

where γ is a constant and Δf the change in the objective function. The algorithm terminates either when the minimum energy is reached or where a predefined (KMAX) number of iterations have elapsed.

2.8 Squeaky Wheel Optimization

The core of Squeaky Wheel Optimization (SWO) framework [63] is a Construct/Analyze/Prioritize cycle, as shown in Figure 2-6(a). Constraint variables are placed in a priority queue in decreasing order of an initial estimate of the difficulty to assign a value to each one of them. A solution is constructed by a greedy algorithm, taking decisions in the order determined by the priority queue. The solution is then analyzed to find those variables that were the "trouble makers." The priorities of the "trouble makers" are increased, causing the greedy constructor to deal with them sooner in the next iteration. This 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 2-6(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 [115]. SWO was originally adapted to an earlier version [101] of the problem formulated in Section 1.1. This is the version that is used by SelfPlanner 1.

The greedy construction algorithm defines a function g from the priority queues to the solutions, that is, for each ordering p of the activities, a schedule $g(p)$ is defined. However, function g may be neither surjective nor injective; thus, many feasible solutions may not correspond to any ordering of the activities in the queue.

The adaption uses several domain-dependent heuristics that measure the impact of the various ways of scheduling a specific activity (including both time and location) to the remaining ones. In particular, the difficulty $\mathtt{diff}(T_i)$ to schedule an activity T_i is defined as the maximum between two metrics, m_1 and m_2, which in turn are defined as follows:

Metric m_1 of an activity T_i is defined as the ratio between the total duration of the activity and the net size of its temporal domain, i.e.,

$$m_1(T_i) = \frac{d_i}{\mathtt{net}(D_i)}$$

where the net size $\mathtt{net}(D)$ of a temporal domain D consisting of a set of intervals is defined as the sum of the widths of these intervals. Metric m_2 of an activity T_i is defined as the ratio between the minimum possible makespan of the activity and the width of its domain, i.e.,

$$m_2(T_i) = \frac{\mathtt{min}(\mathtt{makespan}(T_i)}{\mathtt{width}(D_i)}$$

The makespan of an activity is defined as the distance between the start time of its earliest scheduled part and the end time of its latest scheduled part. Similarly, the *width* of a temporal domain is defined as the distance between the left end of its leftmost interval and the right end of its rightmost interval. Finally, the overall difficulty to schedule a set of activities S is defined as the product of their individual difficulties,

$$\mathtt{overall}(S) = \prod_{T_i \in S} \mathtt{diff}(T_i)$$

Thus, activities are initially placed in the queue in decreasing order of their individual difficulties, whereas each activity is scheduled at the time slot where the overall difficulty to schedule the remaining activities is minimized. For each possible time window to schedule the current activity, constraint propagation is employed to update the temporal domains of the remaining activities before computing their difficulties.

In case of preferences, a ratio between overall difficulty and approximated overall utility is minimized,

$$\frac{(overall(S))^a}{\left(\sum_{v_k \in V} v_k\right)^b}$$

The two key features of the implementation of SWO are worth mentioning. First, for each alternative option to schedule the current activity, constraint propagation is employed before evaluating the option. In this way, infeasible scheduling options and failures are detected earlier. Second, troublemakers are promoted aggressively at the top of the priority queue. It has been found empirically that both constraint propagation and promotion to the head of the priority queue result in significant reduction in the number of iterations and the time needed to solve a problem.

Finally, it has been shown experimentally in Refanidis (2007) [101] that for this problem SWO with the difficulty heuristics is usually more efficient and effective (under time limit) than a complete constraint propagation search algorithm with the usual general heuristics. Thus, taking into account that the scheduling problems have many degrees of freedom (i.e., many activities, large temporal domains, alternative ways to split interruptible activities, alternative locations for each part of an activity, etc.), whereas all problems are solved centrally, sacrificing completeness in favor of an efficient and predictable scheduling algorithm is in our view a justified decision.

SWO was later adapted to the Constraint Optimization Problem presented in Section 1.1 of this thesis [105]. 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 (1.5).

CHAPTER 3

Considering Various Methods for Optimization

The Squeaky Wheel Optimization Framework (SWO) scheduler served as our benchmark, both for solution quality and algorithm efficiency, as we explored various methods to produce better-quality plans. The motivation for this was based on the observation that in many schedules produced by SWO, a user could easily improve upon those plans by shifting some scheduled activity parts temporally in their respective domains. Any scheduler designed and implemented should pass the time-restrictions required for its use in an electronic calendar application. We define those requirements as a run time of up to 30 seconds for the production of a valid high-quality schedule. Optionally, the user should be able to choose if he prefers having a poorer quality valid plan calculated in a short amount of time or a better quality one that takes more time to produce. SWO proved to be a very fast algorithm for the problem described, producing solutions ranging in time from some milliseconds to 42 seconds of CPU time (in extremely complicated problem instances).[1]

The rest of this chapter is structured as follows: We first consider modeling and solving the Constraint Optimization Problem of Section 1.1 using Constraint Logic Programming. We explore the problem space by testing custom heuristics and by using local search. Subsequently we consider a solver based on Genetic Algorithms for the problem. In addition, we extend the problem formulation by adding support for non-monotonic temporal utilities. The insights gained from the local search method lead us to the modified Simulated Annealing algorithm, empowered with Tabu Lists and backtracking and coupled with in-domain heuristics that is presented from Section 3.4 onwards. To describe our proposed method, we first present the local search transformations to explore the neighborhood during the post-optimization phase, using either *hill-climbing* or *Simulated Annealing* [61, 70]. Subsequently, we present experimental results over a large set of problem instances, taken from the literature [105]. Furthermore,

[1] All CPU time measurements conducted in this thesis were run on a dual Intel Xeon X5650 2.66GHz computer with 16GB of 1333MHz DDR3 ECC RAM running Darwin OS. All schedulers tested and implemented were compiled as 64-bit binaries using gcc/g++ with the -O3 compiler optimization flag (with the exception of the schedulers in ECL^{i}PS^e Prolog, which were tested using the 64-bit version of ECL^{i}PS^e).

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 chapter and identifying directions of future work.

3.1 Constraint Logic Programming

We modeled the Constraint Optimization Problem of Section 1.1 using ECL^{i}PS^e Prolog's IC (Interval Constraint) library, a general interval propagation solver that supports problems over both integer and real variables. All the decision variables of the problem are defined over the integer domain (these being p_i, t_{ij}, d_{ij} and l_{ij}, for each T_i and T_{ij}), while a lot of auxiliary variables over the real domain are required to model the problem's objective function.

In our model[2] of the problem we define the following decision variables and their respective domains:

$\forall t_{ij}$: `T#::[-1|Domain]`

$\forall d_{ij}$: `D#::[0,Smin..Smax]`

$\forall l_{ij}$: `Loc#::[-1|Locs]`

where `Domain`, `Smin/Smax` and `Locs` refer to D_i,[3] $smin_i/smax_i$ and Loc_i of each activity T_i. An assignment to the above decision variables, defines an assignment to an activity part (T_{ij}). We set $p_i = 1$ for non-interruptible activities and for interruptible activities we set p_i to the maximum number of activity parts that an interruptible activity can be scheduled in (p_i^{max}):

$$p_i^{max} = \left\lfloor \frac{d_i^{max}}{smin_i} \right\rfloor \tag{3.1}$$

For each activity part T_{ij} not included in the schedule, the part is constrained to $t_{ij} = -1$, $d_{ij} = 0$, $l_{ij} = -1$. To order the scheduled activity parts in ascending order (according to their start times t_{ij}), and place the unscheduled parts in the end, we add the following constraints:

```
T#=-1    ⟹    D#=0
D#=0     ⟹    T#=-1
D#=0     ⟹    Loc#=-1
PT#=-1   ⟹    T#=-1
PT#⩾0    ⟹    T#⩾PT or T#=-1
```

[2]The whole source code of our model is included in Appendix C.

[3]`Domain` refers to a filtered version of D_i, where for every interval $[a_{ik}, b_{ik})$ we use $[a_{ik}, b_{ik} - smin_i)$ (where k is the number of intervals of D_i).

where PT refers to previous activity part t_{ij-1}. For each activity we add the Flag decision variable that takes a value of 0 when the activity is not included in the schedule and 1 if it is included. To include an activity in the schedule the following constraints must hold:

```
Flag#=1  ⟹  Duration[4]#⩾DurMin
Flag#=1  ⟹  Duration#⩽DurMax
```

Otherwise every activity part is constrained to $\{-1, 0, -1\}$ and thus the activity is omitted from the schedule. For modeling location distances we add the following constraints between every pair of activity parts, either of the same activity or belonging to different activities:

```
Flag#=1 and D#>0 and D2#>0                ⟹ (T+D+LocDist)#⩽T2
Flag#=1 and Flag2#=1 and D#>0 and D2#>0   ⟹ (T+D+LocDist)#⩽T2 or (T2+D2+LocDist)#⩽T
```

where LocDist is the distance between the activity parts' scheduled locations. An activity can be scheduled in any location belonging to its Loc_i set, thus LocDist has a different value depending on the l_{ij} values of the two activity parts where the above constraint is defined over. We use the Propia library's *infers most* constraint, that transforms any Prolog *Goal* into a constraint, to model this.

To model the utilization constraints[5] we add auxiliary variables for all the time points in the domain that have a value equal to the sum of all utilization values of all overlapping activities scheduled at that time point. These variables are constrained not to have a value exceeding the unit (i.e., if an activity has $utilization_i = 1$, meaning that it cannot be overlapped, that limit is reached for the time points where the said activity is scheduled).

We also add constraints between parts of both the same activity (unary constraints) and parts belonging to different activities (binary constraints) to provide:

- Binary ordering constraints between the activity parts scheduled (i.e., activity A's parts should be scheduled before activity B's parts).
- Unary and binary *minimum* distance proximity constraints that specify the minimum distance allowed between activity parts.
- Unary and binary *maximum* distance proximity constraints that specify the maximum distance allowed between activity parts.
- Binary implication constraints that imply for an activity A to be included in the schedule, activity B should also be included (Flag1#=1 ⟹ Flag2#=1).

[4] Duration is an auxiliary variable defined as $\sum_{j=1}^{p_i} d_{ij}$.

[5] The model allows overlapping of different activities at the same time, provided that the sum of said activities utilization values does not exceed the unit.

To define the objective function that attempts to model all the user preferences (Formula 1.5), we use auxiliary variables over the real domain to model the following objectives that provide the sources of a plan's utility:

- Unary temporal preferences that support four monotonic functions (linear ascending, linear descending, step ascending, step descending).
- Activity utility (according to Formula 1.1).
- Unary minimum and maximum proximity preferences (Formulas 1.2–1.3).
- Binary ordering, minimum/maximum proximity (Formula 1.4) and implication preferences.

By using ECL^{i}PS^e's branch and bound algorithm we can solve instances of the problem. While this is an *exact* method that defines a complete search algorithm, we could obtain optimal solutions only for trivial problem instances. In anything more complicated the search space exploded and we terminated the algorithm after one to two days of run time, where the best solution found thus far had a lower utility than SWO's solution. For example on the 5-activity p5_1 problem instance, taken from the literature [105], SWO gives a solution of 83.5973 utility out of a loose upper bound of 84, while the best solution found before being terminated from ECL^{i}PS^e's branch and bound solver is 47.4707. In even more complicated problem instances (e.g., p30_1) the solver required approximately 7 minutes and 20 seconds of CPU time before even the branch and bound algorithm started finding solutions. We also evaluated other search methods, besides complete search, such as bounded backtracking search, limited discrepancy search and credit based search with no more success. In addition, we tried various variable selection heuristics and various value ordering heuristics. Our best results seemed to be while using the *most constrained* variable selection heuristic and the *in-domain split* value ordering heuristic.

3.1.1 Defining Custom Heuristics

Another option would be to search through the space of possible repairs using a min-conflicts heuristic. Minton et al [81] reported success in solving large scale constraint satisfaction and scheduling problems with this approach—though they tested their method on CSPs, not Constraint Optimization Problems. Indeed, repairing all the conflicts before attempting to optimize could lead the search to inefficient solutions, as the algorithm would not take the hit of moving the search to new constraint violations before reaching better assignments for the decision variables that are valid.

Using ECL^{i}PS^e's repair library we can set *tentative* values and associate them with any variable. This allows us to violate constraints on the variable, while keeping a record of all constraint violations. Moreover, changes to a variable's tentative value are propagated to other variables whose tentative values depend on the changed one. We can use the repair library to search

```
feasible_durations(_Flag,[],_DurMin,_DurMax,_Smin,_Smax,_Duration).
feasible_durations(Flag,[subtask(_T,D,_Loc)|L],DurMin,DurMax,Smin,Smax,Duration):-
    Cnt is Smax-Smin,
    feasible_durations1(Flag,D,DurMin,DurMax,Smin,Duration,Cnt),
    feasible_durations(Flag,L,DurMin,DurMax,Smin,Smax,Duration).

feasible_durations1(_Flag,_D,_DurMin,_DurMax,_Smin,_Duration,-1).
feasible_durations1(Flag,D,DurMin,DurMax,Smin,Duration,Cnt):-
    A#=DurMin-Duration-(Smin+Cnt),
    B#=DurMax-Duration-(Smin+Cnt),
    K1F$=(A-1)/(Smin+Cnt),
    floor(K1F,K1) infers most,
    K2F$=(B+1)/Smin,
    ceiling(K2F,K2) infers most,
    (Flag#=1 and A#>=0 and K2$=K1+1 and Cnt#>0) => D#\=(Smin+Cnt),
    Cnt1 is Cnt-1,
    feasible_durations1(Flag,D,DurMin,DurMax,Smin,Duration,Cnt1).
```

Figure 3-1: Feasible durations heuristic

either through the space of possible repairs (and thus follow Minton's approach) or search the whole space of variable assignments (including invalid solutions). Of course the second space is huge especially while considering the failure of the complete search method on the space of valid solutions.

Before attempting to use repair-based search methods, we applied custom heuristics to guide the search using in-domain knowledge of the problem. Particularly, we wanted the branch and bound algorithm to consider assigning a value to a whole activity part T_{ij} at once, that is to assign values for that part's t_{ij}, d_{ij} and l_{ij} at every consideration. To achieve this goal we designed our own variable selection and value ordering heuristics.

We tried various ways of ordering the assignments of every part's t_{ij}, d_{ij} and l_{ij} variables, without improving the success of the standard heuristics. In fact the performance of the solver was even worse, finding poorer solutions and producing a timeout for the p5_1 problem instance before it found even one solution.

We also designed the feasible durations heuristic which utilizes auxiliary variables and constraints to prune the domain of every decision variable d_{ij}, which corresponds to the activity parts' durations, from evaluating non-possible assignments. This heuristic monitors the current value of $\sum_{j=1}^{p_i} d_{ij}$, for each activity T_i, to prune the domain of each d_{ij}.

3.1.2 Applying Local Search

We replaced the branch and bound search algorithm with a local-search algorithm we implemented in ECL^{i}PS^e's that searches the space of all possible assignments to the decision variables guided by the list of possible repairs. For each assignment of the decision variables the *conflict sets* of the variables, that violate the constraints of the problem, are stored and the search

is guided by various heuristics. The problem was effectively converted to an unconstrained optimization problem (though the variables were still constrained to be inside their domain), where each constraint violation had a huge cost to the objective function.

The local-search algorithm implemented was initially based on *hill-climbing*, but subsequently expanded to utilize a *Tabu List*. The initial assignment of the decision variables was random (inside their domain) and the search was guided by various heuristics that attempted to utilize in-domain characteristics of the scheduling problem formulated as a Constraint Optimization Problem. Particularly, we assigned values to whole activity part T_{ij} at each consideration (t_{ij}, d_{ij}, l_{ij}) and the search was guided by the $feasible\ duration$ domain-pruning heuristic.

The algorithm gave us some encouraging results in simple problem instances, but did not terminate in harder ones taken from the literature [105] (even in the $p5\_1$ problem instance). The new search-space that included invalid solutions proved to be too large for this method. Based on the insights received from this approach we concluded to take the idea of local-search with in-domain heuristics further by:

- Searching only the space of valid solutions as the search space of all possible assignments is huge.
- Define higher-level in-domain heuristics that are based on the problem model to compute the local neighbors of a solution (and thus decrease the search space).
- Recode our model and algorithm in C.

3.2 Genetic Algorithm

A Genetic Algorithm was designed and implemented for a subset of the problem presented in Section 1.1. Our goal for the Genetic Algorithm was to reach competitive results to SWO and afterwards expand the algorithm to solve instance of the full problem. The algorithm was implemented in the C programming language [65]. The algorithm operates as follows:

1. Create an initial population of chromosomes with the following characteristics:

   a. Each chromosome of the population corresponds to a set of activities which represent a plan π_i for a problem instance.

   b. Each such chromosome is represented by a matrix $\sum_{i=1}^{N} p_i^{max} \times 3$,[6] where p_i^{max} is calculated using Formula 3.1. This matrix holds the decision variables for every activity part T_{ij} of every activity T_i.[7]

2. An initial solution is found for each activity of every chromosome by the $insertTask$ algorithm. This algorithm takes as input a chromosome and the index i of an activity to

[6] In the relaxed problem it is $\sum_{i=1}^{N} p_i^{max} \times 2$ as locations were omitted.

[7] The decision variables p_i are excepted as their value will be decided from the number of activity parts having $t_{ij} > -1$ (of each activity T_i and $1 \leqslant j \leqslant p_i^{max}$).

be scheduled and attempts to add activity parts until the required total duration ($d_i^{min} \leqslant \sum_{j=1}^{p_i} d_{ij} \leqslant d_i^{max}$) of an activity is reached. It works as follows:

a. A tree structure is created, where the root node is the chromosome.

b. Every part T_{ij} of activity T_i is scheduled stochastically, respecting all the constraints, until the required total duration for T_i is met. This is accomplished by expanding the search tree with new nodes for each possible assignment of the decision variables (out of their domain), for a given i and j. The algorithm conducts a depth-first search.

c. If the maximum duration of the activity (d_i^{max}) is passed, the algorithm backtracks stochastically using a probability distribution that favors older nodes (i.e., nodes closer to the root).

3. Execution of the genetic algorithm commences.

   a. Formula 1.5 acts as the *fitness* function, which also performs constraint propagation.

   b. The *crossover* operation combines half activities of one chromosome, with half of the other.

   c. The *mutation* operation alters stochastically 0 to x activity parts of a chromosome. For altering those parts, the algorithm removes them completely and subsequently re-adds them by running the $\mathtt{insertTask}$ algorithm.

4. The selection of chromosomes for crossover is decided by a probability distribution that gives higher priority to higher fitted chromosomes. Also, again stochastically with the probability distribution, the best chromosomes pass to the next generation intact, while the worst neither do they cross-over nor do they pass to the next generation.

We compared the Genetic Algorithm (GA) to SWO on a 60-activity problem instance taken out of the $p60$ problem instances, that was relaxed to the implemented subset of the problem model in the GA. The problem instance used for the evaluation had a *loose upper bound* of $1,143$ The results of the comparison are presented in Table 3.1.

Table 3.1: GA vs SWO

Algorithm	Plan Utility	CPU time (in secs)
GA	494.5674	$1,762.0659$
SWO	906.651	39.1753

3.3 Non-Monotonic Temporal Utilities

We expanded the problem model (Section 1.1) by redefining the $U_i^{time}(\pi_i)$ function to support non-monotonic temporal utilities. The new $U_i^{time}(\pi_i)$ function allows us to model non-

monotonic temporal preferences, such as the preference of an activity's part to be scheduled in the morning, near noon, at the afternoon, at specific days etc. Non-monotonic temporal utilities are defined as follows:

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:

$$U_i^{time}(\pi_i) = \frac{\sum_{j=1}^{p_i} \sum_{t=t_{ij}}^{t_{ij}+d_{ij}-1} u_i^{time}(t)}{d_i} \tag{3.2}$$

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, $a_{ik} \leqslant t < b_{ik}$, $1 \leqslant k \leqslant F_i$. 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:

$$u_i^{time}(t) = \begin{cases} 0, \\ \text{ if } t < a_{ik} \text{ or } t \geqslant b_{ik}, \text{ for every } k \\ u_{ik}^{low} + (t - a_{ik})\frac{u_{ik}^{high} - u_{ik}^{low}}{b_{ik} - a_{ik} - 1}, \\ \text{ if } t \geqslant a_{ik} \text{ and } t < b_{ik} \text{ for some } k \end{cases} \tag{3.3}$$

Any non-monotonic function can be used to map each a_{ik} and b_{ik}, of an activity's T_i domain, to u_{ik}^{low} and u_{ik}^{high} according to the user's temporal preferences for that activity.

3.4 Optimizing Individual Activity Personal Plans through Local Search

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 [9, 13, 41, 85, 96, 101, 102].

In Section 1.1, 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. The scheduler, based on the Squeaky Wheel Optimization (SWO) framework and coupled with domain-dependent heuristics, which was discussed in the second chapter of this thesis (Section 2.8), 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 [28, 33, 74, 87, 112]. So, in the following sections we present local-search techniques that increase the quality of SWO's plan output through 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 [2, 4]. The following sections elaborate over previous work of the same authors in that (a) they advance 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 over-constrained problems prior to local search; (b) they provide detailed experimental results, including evaluating the contribution of each individual transformation to the final output; (c) they illustrate the effectiveness of the proposed local search optimization algorithms through a realistic scenario; and (d) they discuss usability and transferability issues.

3.5 Applying Local Search Methods to Improve SWO Output

In previous work [2] 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 [4] 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.

```
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
```

Figure 3-2: Hill-climbing based post-processing algorithm

Figure 3-2 gives an overview of HC, as it has been applied to the problem of optimizing individual activity personal plans. U denotes the objective function (Formula 1.5) 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*.

3.5.1 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 to 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 $d_{ij} + d_{ik} \leqslant smax_i$ 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.

Transferring 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 $d_{ij} \geqslant 2 \times smin_i$, 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 $d_i^{min} \leqslant d_i < d_i^{max}$, it is attempted to add a new part in the current schedule, provided that:

$$d_i \leqslant d_i^{max} - smin_i \tag{3.4}$$

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:

$$\sum_{j=1}^{p_i} d_{ij} \geqslant d_i^{min} \tag{3.5}$$

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}.

3.5.2 Avoiding Local Optima

The *hill-climbing* post-processing algorithm can get stuck in local optima. On the other hand, *Simulated Annealing* [70] 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 3-3) 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 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 3-3 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

```
enhanced-postprocess-swo(Activities, Solution, Best, k, Tabu, schedule, kmax,
emax)
  if k ⩾ kmax  or  U(Best) ⩾ emax
    return Best
  if |Tabu| = kmax/10
    Tabu ← Tabu \ (Oldest_Solution ∈ Tabu)
  Tabu ← Tabu ∪ {Solution}
  t ← schedule[k]
  C ← ∅
  do i ← 0 to ∞
    if i+ k = kmax
        return Best
    New ← random_neighbor(Activities,Solution,C), New ∉ Tabu, (C_ikt ∈ C) ← (C_ikt ∈
C) \ New
    if New = null
        select S out of Tabu stochastically favoring older solutions
        return enhanced-post-process-swo(Activities, S, Best, k, Tabu \ S, schedule, kmax,
emax)
    ΔE ← (U(New)−U(Solution))/e
  until ΔE > 0  or  select New with probability e^(ΔE/t)
  k ← k + i + 1
  if New > Best
    Best ← New
  return enhanced-post-process-swo(Activities, New, Best, k, Tabu, schedule, kmax,
emax)
end
```

Figure 3-3: Simulated Annealing based post-processing algorithm

of |Tabu| (the number of past states kept in the Tabu List) to $\frac{kmax}{10}$. These values were chosen after extensive experimentation.

3.6 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.

3.6.1 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 the literature [105]. 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 [105], 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 3-4. 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 3-4, 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 3-5 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 3-4. Having in mind real-world situations, we consider the time requirements acceptable, since the problems of the test set are artificially created with increased

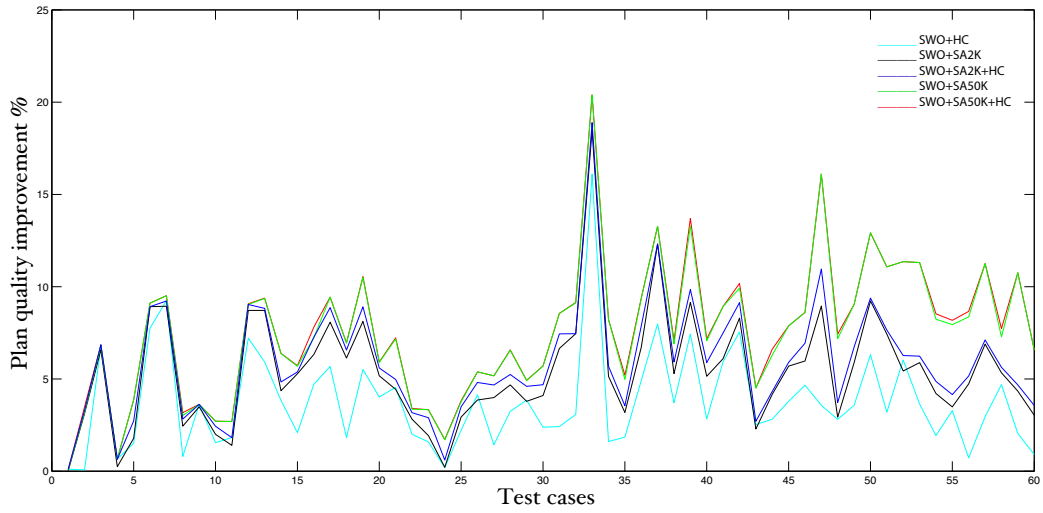

Figure 3-4: Improvement of plan quality over SWO

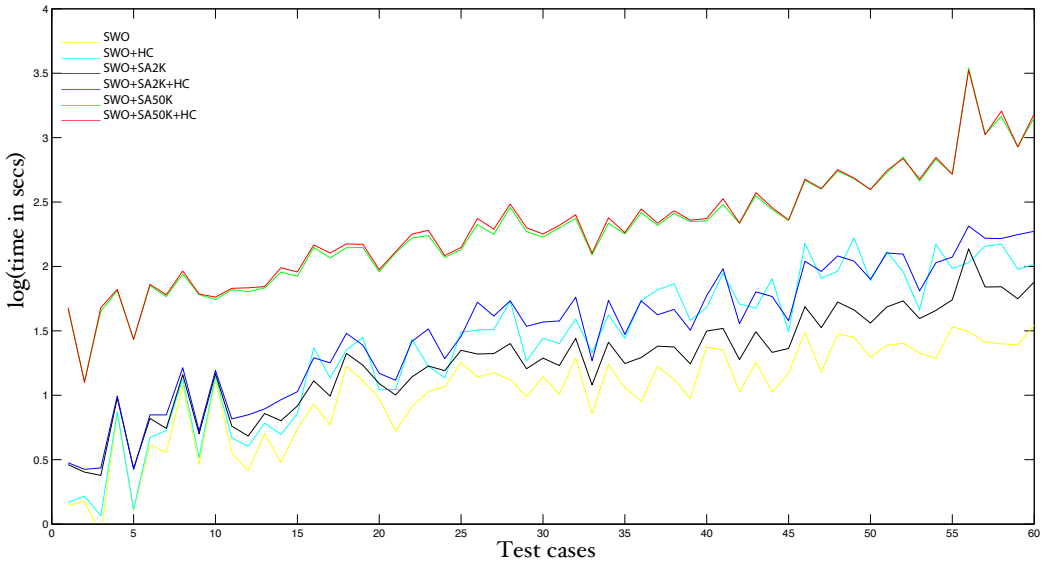

Figure 3-5: Execution time results

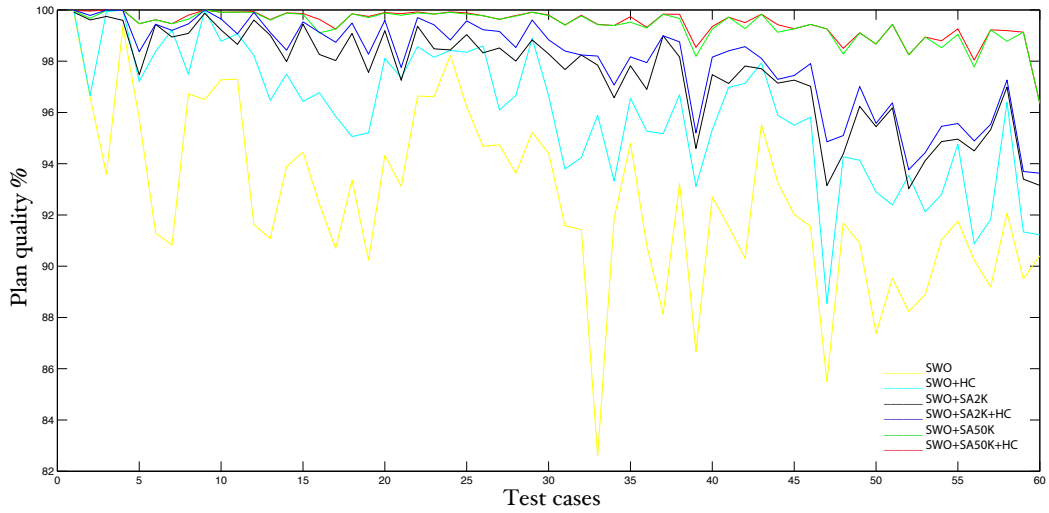

Figure 3-6: Quality relative to the *experimental upper bound*

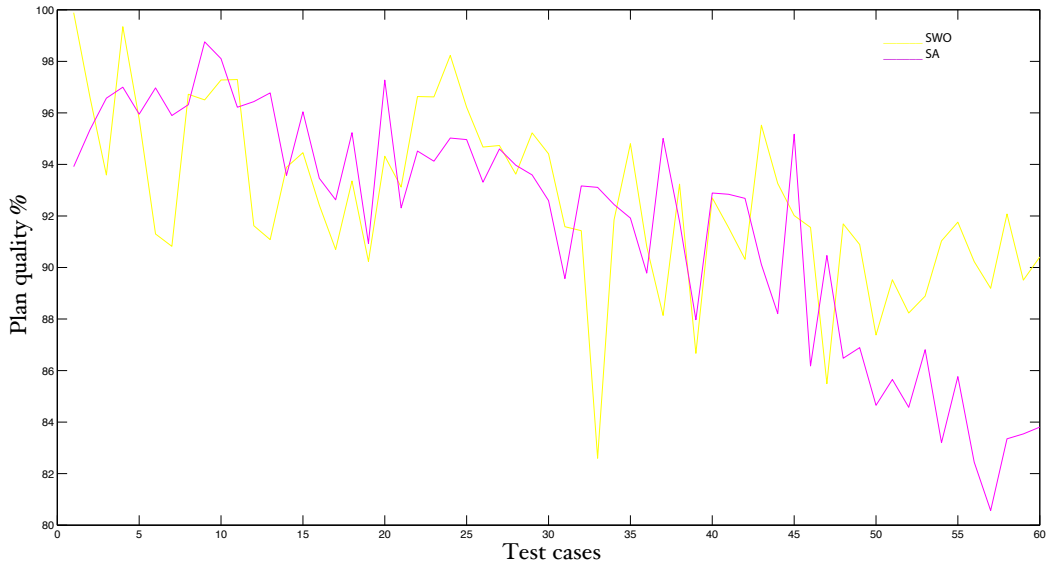

Figure 3-7: Running SA from the empty plan for the same time as SWO. Quality relative to the *experimental upper bound*

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 run-times, 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 3-6. 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.

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 literature [105]. The results, compared to the experimental upper bound, are shown in Figure 3-7. 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 the literature) 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 2-6b). 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.

3.6.2 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

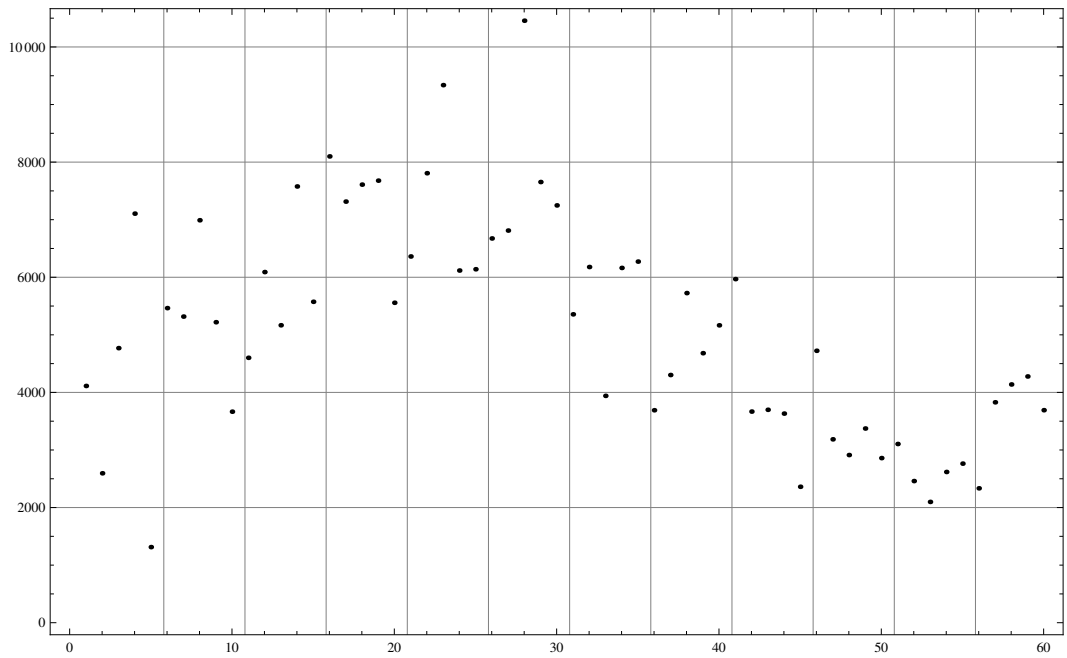

Figure 3-8: Average neighborhood size for each test case

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 3-8. 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.

3.6.3 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 3-9 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 omitted 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

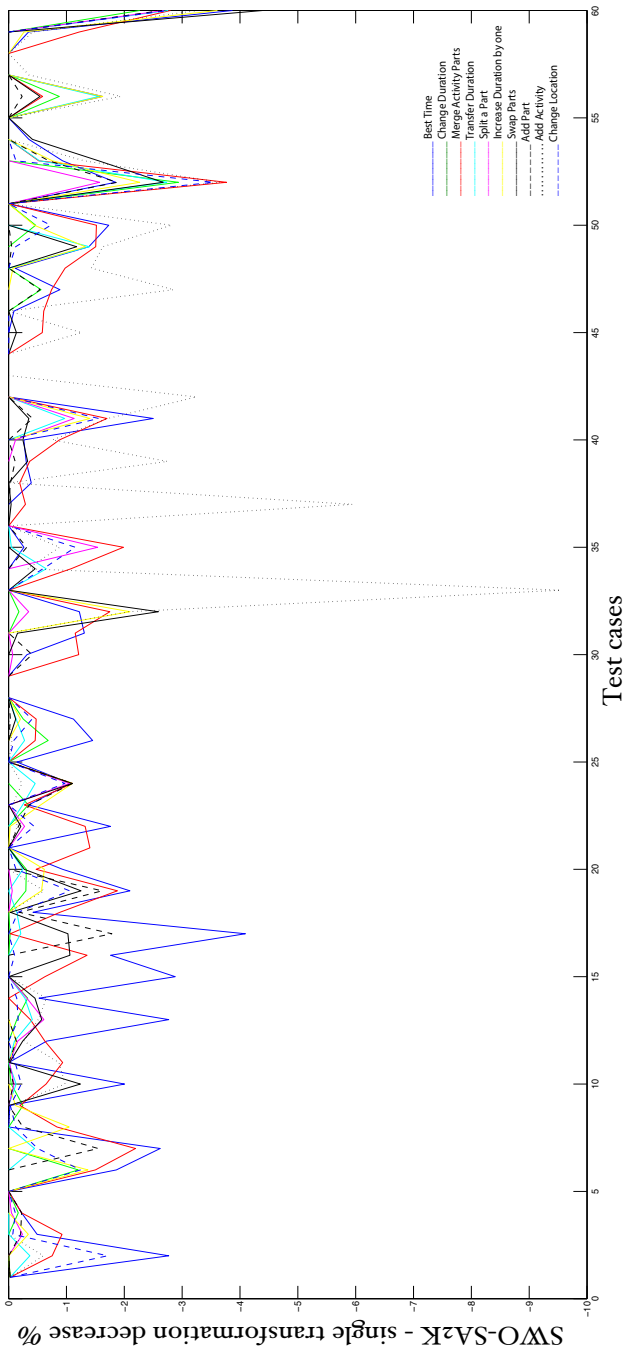

Figure 3-9: Percentage decrease in the quality of the resulting plans when nine out of the 10 transformations are employed. Each line corresponds to removing a single transformation from their bundle.

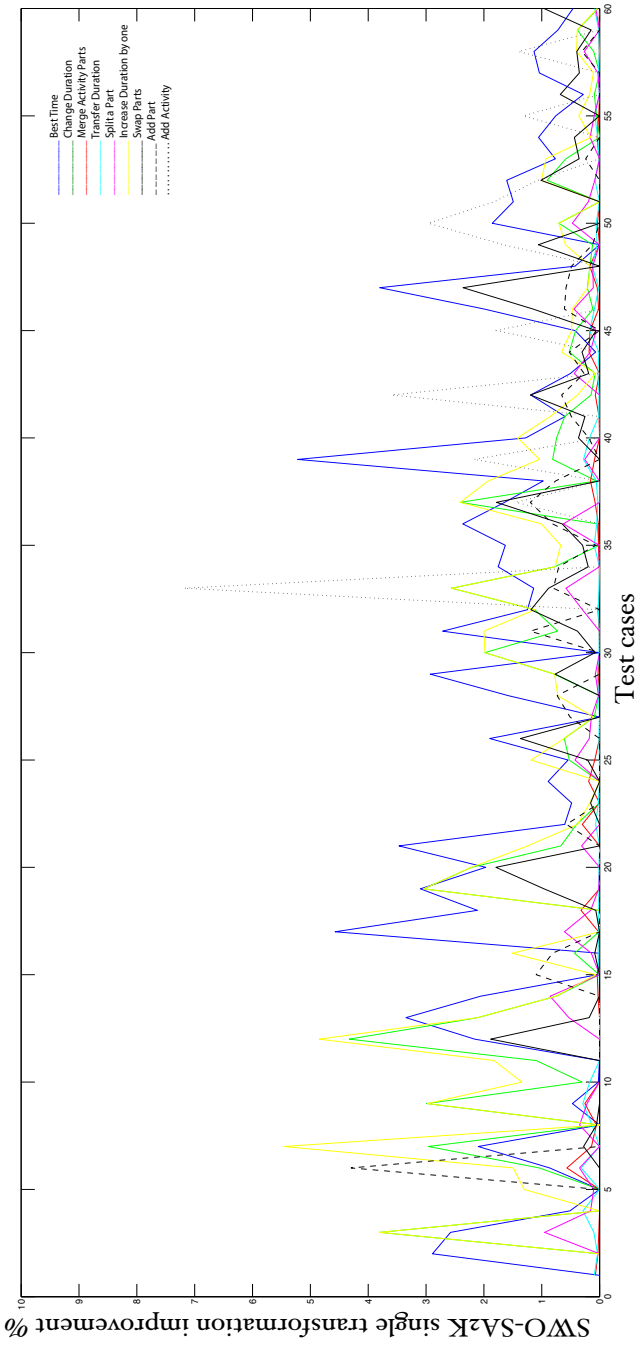

Figure 3-10: Percentage increase in the quality of the resulting plans when each transformation is employed separately. Each line corresponds to a single transformation being employed.

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 3-10 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 $U(\mathtt{time})$ 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.

3.6.4 A Realistic Scenario

In this section we compare SWO+SA50K against SWO on a realistic scenario. The motivation is two-fold: 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

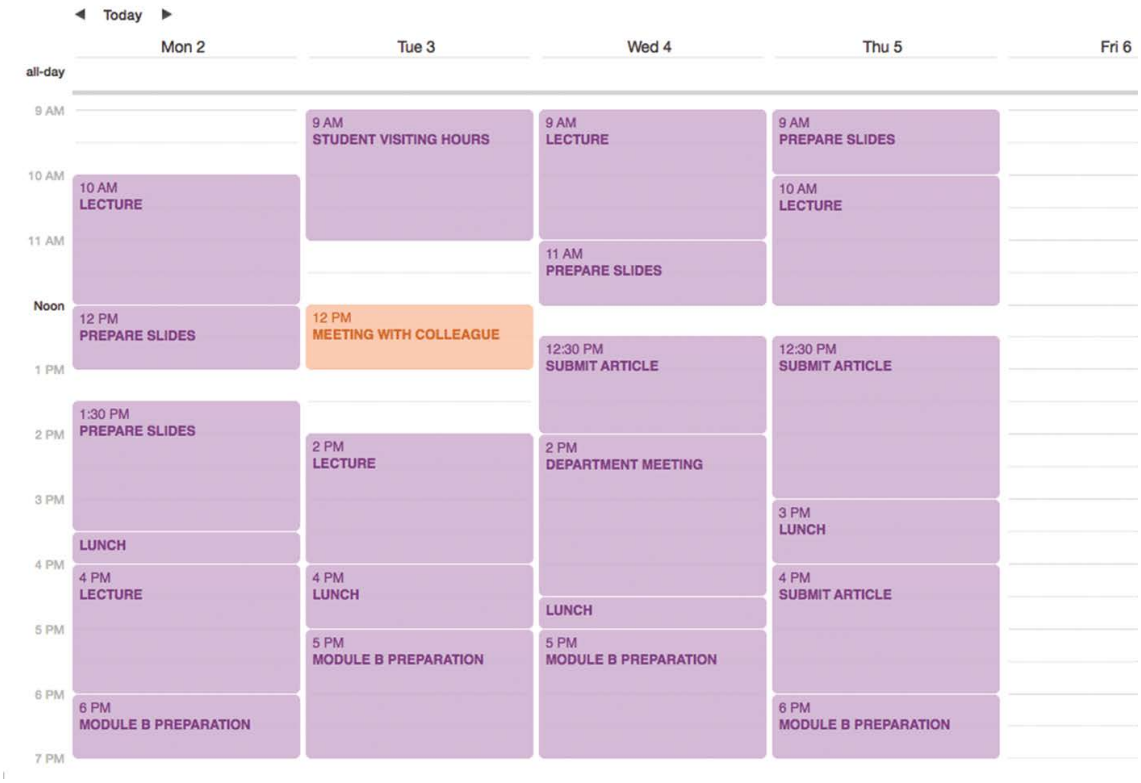

Figure 3-11: SWO's solution

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.

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

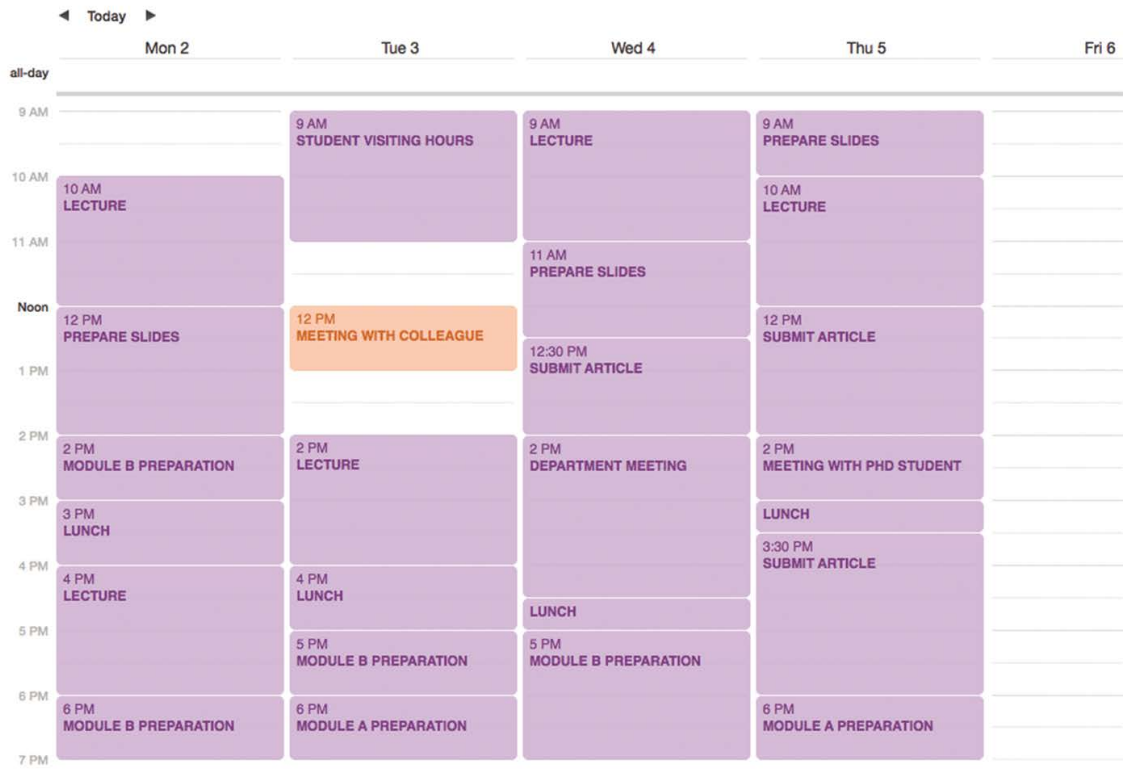

Figure 3-12: SWO+SA50K's solution

(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 pre-determined 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 3-11 and 3-12. Since the post-processor is a stochastic algorithm, we executed it 100 times on Ann's problem and selected to present in Figure 3-12 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 3-11) 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 3-11 as its seed solution, SWO+SA50K produced the schedule shown in Figure 3-12. 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 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 "Submit article" gives the highest utility per unit of extra duration, so both schedules included it with its maximum duration (6 hours). In Figure 3-12 we can observe that the allocated duration for "Prepare slides" was decreased by half-an-hour compared to Figure 3-11,

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 3-11, 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 3-12 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 chapter.

3.6.5 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 3.2: 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 3.2 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.

3.6.6 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.

3.7 Summary and Future Work

In this chapter, after experimenting with various other methods, 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 chapter, 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 chapter. Using heuristics to select a subset of the transformations to consider might be of great significance, in order to keep the whole approach efficient.

CHAPTER 4

Multiple Plans & Machine Learning

In the previous chapter we proposed an algorithm for producing optimized solutions. The algorithm can either be run from the empty plan (see Figure 3-7) or can use SWO's solution as the seed solution. In this chapter we extend both the model and the algorithm to produce multiple solutions for a given problem instance. We also introduce a machine learning method that attempts to learn the user's qualitative preferences.

4.1 Introduction

SelfPlanner [102] is a prototype application designed for the automatic scheduling of a user's individual activities in an electronic calendar. It supports a rich domain model for scheduling interruptible/non-interruptible activities, as well as periodic activities with various unary and binary constraints and preferences that can be defined on them by the user.

In this chapter we extend our previous work on scheduling individual activities by introducing new methods to generate qualitative, significantly different alternative plans. In this way we increase the chances that at least one of them meets the user's preferences. According to Kambhampati [64], in many real world planning scenarios the user's preferences are either unknown or at best partially specified. This renders increasingly complex to capture the user's preferences in a personal activities scheduling problem. Some systems attempt to elicit user preferences in a non-intrusive manner, by presenting alternative plans to the user and building a preference model based on his choices [13, 85].

Intelligent assistance for time and task management has been targeted by previous AI research [85, 41, 101, 102, 13, 9]. In common electronic calendar applications, schedules comprise sets of fully specified and independent events. An event is characterized by a fixed start-time, a duration, and optionally a location. Furthermore, many systems support tasks. These represent individual commitments potentially having a deadline to be met (e.g., writing an article or doing homework). Tasks are usually kept in separate task lists and do not have a specific start time. Converting a task to an event is usually a straightforward procedure that is accom-

plished by dropping the task into the electronic calendar.

SelfPlanner is based on a domain model that treats events and tasks in a uniform way. They are called *activities* and are characterized by a number of attributes, such as a temporal domain, a duration range, a set of alternative locations, as well as unary constraints and preferences over the ways the parts of an interruptible activity are scheduled in time. In addition, the model also supports binary constraints and preferences (ordering, proximity and implication) between pairs of activities. The model specifies a Constraint Optimization Problem (COP), where the best solution is the one that maximizes the total utility of the plan, linearly aggregating the various sources of utility. SelfPlanner's scheduler was presented in the previous chapter.

In the literature one can find two general approaches to generate alternative plans with a planner or scheduler. The first approach is based on managing either the set of possible solutions or the order in which they are evaluated by the search algorithm [107, 36]. The second one—which we based our alternative plan generation method on—is based on modifying the heuristic of the planning algorithm [29, 92, 86].

In this chapter we present our approach to generate qualitative, significantly different alternative plans, based on the enhancement of the plan evaluation function used by the scheduler during the optimization process with additional attributes that measure the difference to the already generated plans. We define a plan difference function that utilizes domain-specific traits of the problem formulation. In addition, we introduce a machine learning method to forecast and compensate systematic errors performed by the user when defining his preferences over the various aspects of a plan. This chapter extends our previous work [3], by introducing machine learning in the generation of alternative plans; by presenting fresh experimental results in all cases; by adopting a system oriented view, than a pure algorithmic one; and, finally, by presenting a more extensive related work and discussion of the results.

The rest of this chapter is structured as follows: Section 4.2 focuses on the hybrid evaluation function that considers (among other criteria) differences to already generated plans. This is based on the introduction of a plan difference function to measure the distance between pairs of plans. Section 4.3 employs machine learning to reveal and compensate systematic errors in expressing a user's preferences, based on his choices over the alternative plans presented to him. Section 4.4 demonstrates multiple plan generation on a number of randomly generated problem instances. In addition, it simulates two virtual users' preferences and evaluates the machine learning method. Finally, Section 4.5 concludes the chapter and identifies potential directions for future work.

4.2 Generating Alternative Plans

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

4.2.1 Hybrid Evaluation Function

In order to generate an arbitrary number of qualitative, significantly different alternative plans, we extended the domain model by introducing the notion of the *value* of an alternative plan. This is defined as a linear combination of pure utility, as defined by Formula (1.5), with the deviation of the plan under evaluation from the already generated plans. Let Δ be the set of the already generated plans, the value of a new plan π is defined by Formula (4.1), as a replacement to Formula (1.5):

$$V(\pi) = U(\pi) + C \times \frac{\sum_{\forall \pi' \in \Delta} U(\pi') \times \mathtt{PDiff}(\pi, \pi')}{max(|\Delta|, 1)} \tag{4.1}$$

where C is a constant that weights the importance of π's difference over Δ's plans versus its utility, and $\mathtt{PDiff}(\pi, \pi')$ is a function assessing the differences of any two plans π and π'. As it will be shown later in this section, $\mathtt{PDiff}(\pi, \pi')$ ranges between 0 and 1, with higher values denoting greater differences between the two plans; $\mathtt{PDiff}(\pi, \pi') = 0$ denotes no difference. By including factor $U(\pi')$ in the product in the numerator of Formula (4.1), new plans deviating from already generated high utility plans are favored over new plans deviating from already generated lower quality ones. Furthermore, the factor of $U(\pi')$ scales the second term of the sum to the scale of the first one. $|\Delta|$ in the denominator of Formula (4.1) refers to the number of the already generated plans.

In case Δ is empty, that is no plan is already generated, Formula (4.1) reduces to Formula (1.5). Having found a nonempty set of plans, the scheduler tries to maximize concurrently the utility of the new plan, $U(\pi)$, as well as its average deviation from the already found plans, $U(\pi') \times \mathtt{PDiff}(\pi, \pi')$, weighted by constant C.

Note finally that the hybrid evaluation function (4.1) is used only during the scheduling process. The generated plans are presented to the user ordered according to their pure utility, that is, $U(\pi)$.

4.2.2 Running Example

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

- "Lecture" has to be scheduled on Wednesday evening, at 15:00, with a fixed duration of 2h, at the University.
- "Visit Library" has a variable duration between 1 and 3 hours, the library visiting hours (for the whole week) as its temporal domain, with the location of the library being downtown.

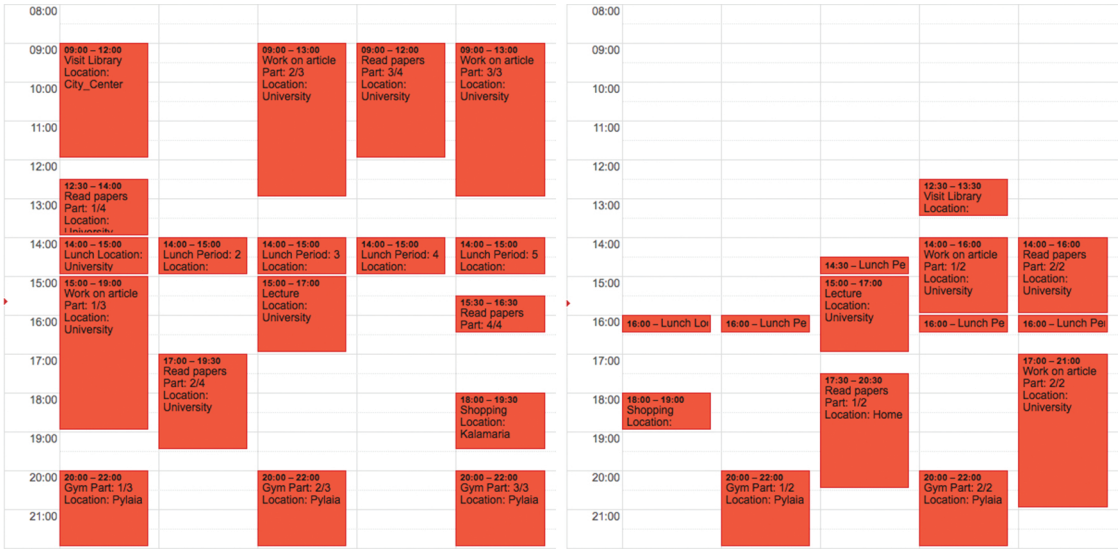

Figure 4-1: (a) The main plan. (b) The alternative plan

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

We set a low value for the utility given to the extra duration of the activities. We defined two constraints: an implication constraint requiring the "Visit Library" activity to be included in the plan if "Work on article" is also included; and an ordering constraint requiring the "Visit Library" activity to be scheduled before all parts of "Work on article."

We used SelfPlanner to produce two alternative plans for the running example, the first being the main plan that the system would normally produce and the second one being gener-

ated using the hybrid evaluation function that takes into account its difference from the first one (Figure 4-1). The C constant of the hybrid evaluation function was set to 1.0. As we have given little utility for the extra duration, the most obvious difference of the two plans is that the second one schedules the user activities closer to their minimum duration. The two plans are as different as they can be, with changes in activity durations, number of activity parts, different locations as well as scheduling times. In the next section, examples are given for the deviation metrics' role in the generation of the alternative plan.

4.2.3 Quantifying the Degree of Deviation Between Two Plans

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

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

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

Durations Deviation

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

$$\Delta D_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \,\vee\, d'_i > 0}} \frac{|d_i - d'_i|}{max(d_i, d'_i)} \tag{4.2}$$

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

Intuitively, the duration deviation measures the degree to which the activities have different durations in the two plans under comparison. In case all activities have the same duration in π

and π', $\Delta D_{\pi,\pi'}$ is 0. In the other extreme, if any activity that is included in π is not included in π' and vice versa, then $\Delta D_{\pi,\pi'}$ is 1.

This metric's role is the easiest to spot on the alternative plan of Figure 4-1(b), since all the activity durations have changed. Since the system managed to schedule all the activities at their maximum duration in the main plan, whereas the utility of the extra duration is kept very low for all activities with variable duration, the alternative plan has all these activities with their minimum duration.

Locations Deviation

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

$$\Delta L_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \,\wedge\, d'_i > 0}} \frac{1}{\min(d_i, d'_i)} \times \sum_{\substack{x=1 \\ \lambda_x^{T_i} \neq \lambda'^{T_i}_x}}^{\min(d_i, d'_i)} 1 \tag{4.3}$$

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

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

This metric's role in the alternative plan in Figure 4-1(b) can be observed on the "Read papers" activity, which can be scheduled either at the University or at home. In the main plan it was scheduled entirely at the University, whereas in the alternative plan one part was scheduled at home and the other parts remained at the University; the reason for that was to avoid unnecessary transfers between them and the adjacently scheduled activities (e.g., "Lunch").

Absolute Time Deviation

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

$$\Delta Time_{\pi,\pi'} = \frac{1}{N} \times \sum_{\substack{T_i \\ d_i > 0 \,\wedge\, d'_i > 0}} \frac{1}{\min(d_i, d'_i)} \times \sum_{x=1}^{\min(d_i, d'_i)} \frac{|\tau_x^{T_i} - \tau'^{T_i}_x|}{width_{D_i}} \tag{4.4}$$

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

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

This metric's role in the alternative plan is very clear (Figure 4-1): No activity was scheduled at the same time it was scheduled in the main plan (except for the "Lecture" activity which had a fixed time).

Ordering Differences

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

$$\nu_{ij} = \frac{1}{d_i \times d_j} \sum_{x=1}^{d_i} \sum_{y=1}^{d_j} \begin{cases} 1 \text{ if } \tau_x^{T_i} \leqslant \tau_y^{T_j} \\ 0 \text{ otherwise} \end{cases} \tag{4.5}$$

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

Furthermore, we extend the definition of ν_{ij} for the cases where both activities are not included in the plan. This is important because otherwise ν_{ij} might be defined in one plan and not in the other. So, if T_i appears in plan π but T_j does not appear in it, we define $\nu_{ij} = 1$. If T_i does not appear in π (irrelevant to whether T_j appears in π or not), we define $\nu_{ij} = 0$. Subsequently, the ordering deviation for a pair of activities, T_i and T_j, in two plans π and π', is defined as:

$$\Delta O_{ij_{\pi,\pi'}} = |\nu_{ij} - \nu'_{ij}| \tag{4.6}$$

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

Finally, we define the total ordering deviation as:

$$\Delta O_{\pi,\pi'} = \frac{2}{N \times (N-1)} \times \sum_{i=1}^{N} \sum_{j=i+1}^{N} |\nu_{ij} - \nu'_{ij}| \tag{4.7}$$

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

This metric's role in the alternative plan in Figure 4-1(b) can observed by comparing the ordering between the activities in the two plans. Most of the orderings have changed, with the exception of the case where an ordering constraint was defined.

4.2.4 Plan Difference Between Two Plans

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

$$\mathtt{PDiff}(\pi, \pi') = \begin{array}{c} \Delta D_{\pi,\pi'} \times W_D + \Delta L_{\pi,\pi'} \times W_L \\ +\Delta \mathtt{Time}_{\pi,\pi'} \times W_{\mathtt{Time}} + \Delta O_{\pi,\pi'} \times W_O \end{array} \tag{4.8}$$

where W_D, W_L, $W_{\mathtt{Time}}$ and W_O are non-negative weights, such as $W_D + W_L + W_{\mathtt{Time}} + W_O = 1$. By specifying the values of these weights, we express our preferences on how we want new plans to deviate from already generated ones.

4.3 Online Learning

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

In the adopted domain model (Section 1.1), utility of a plan arises from several sources, particularly:

1. Activities included in the plan, with their minimum duration.
2. Extra duration for the activities of item (Formula 1.1).
3. The way an activity has been scheduled within its temporal domain.
4. Minimum proximity preferences for the parts of interruptible activities.
5. Maximum proximity preferences for the parts of interruptible activities.
6. Binary ordering preferences.
7. Binary minimum distance preferences.
8. Binary maximum distance preferences.

9. Binary implication preferences.

Defining preferences over all these aspects of a plan is a difficult and error-prone process for the user. So, it is very common for him to overweight some sources of utility and underweight some others. This form of systematic error, that has nothing to do with specific planning problems or activities, is what we attempt to learn and compensate in a non-intrusive manner, by monitoring the choices of the user among the alternative plans presented to him. Compensation is achieved through the introduction of nine compensation factors, named W_1 through W_9, each one of them corresponding to one of the nine aforementioned sources of utility. So, the total utility of a plan is the weighted sum of the utility aggregated from all sources, with the weights being the compensation factors. The initial values of the compensation factors are 1 (in that case, the sum of the various sources of utility coincides to Formula (1.5)), but they are changed by the online learning process, each time the user selects one of the alternative plans presented to him.

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

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

The rationale behind learning the values of W_i is simple: If π' has higher utility than π with respect to utility source i, $1 \leqslant i \leqslant 9$, W_i is increased. If π' has lower utility than π with respect to some utility source i, W_i is decreased. Finally, if π' has the same utility as π with respect to utility source i, W_i is not changed. Furthermore, in order to change each compensation factor proportionately to the contribution of its utility source to the overall utility of the plans, we compute coefficient a_i as the ratio of the contribution of utility source i in π' to the contribution of utility source i in π ($a_i > 1$ implies that W'_i should be increased, whereas $a_i < 1$ implies that W'_i should be decreased). So, the compensation factors are updated according to the following formula:

$$W'_i = \begin{cases} W_i, & \text{if } a_i = 1 \\ \min(W_{max}, W_i \times (1 + Q \times (\alpha_i - 1))), & \text{if } a_i > 1 \\ \max(W_{min}, W_i \times (1 - Q \times (1 - \alpha_i))), & \text{if } a_i < 1 \end{cases} \tag{4.9}$$

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

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

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

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

$$W'_i = \begin{cases} W_i, & \text{if } W_i = 1 \\ W_i - \mu Q \times (W_i - 1), & \text{if } W_i > 1 \\ W_i + \mu Q \times (1 - W_i), & \text{if } W_i < 1 \end{cases} \tag{4.10}$$

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

The presented online learning process is based on the assumption that the error in evaluating the alternative plans is due to wrong compensation of the alternative high level utility sources. However, this is not always the case. For example, it might happen that π is better than π' in all sources of utility, so there is no way to make $U(\pi')$ higher than $U(\pi)$. Such strange cases arise because of errors at a more fine-grained level, that is, in compensating utility sources of the same type. As an example, consider the first source of utility, that is, the activities themselves. Suppose that the user assigns more utility to T_1 than to T_2, although he prefers mostly T_2 to T_1. This type of error cannot be learnt by the proposed approach. However, this type of error is not a systematic error, since in the next scheduling problem, T_1 and T_2 will probably not be present. Such errors could be considered systematic by the introduction of ontologies. Indeed, if T_1 is an instance of class C_1, T_2 is an instance of class C_2 and the user systematically overestimates activities of class C_1 to activities of class C_2, this could be learnt by an online learning algorithm and compensated using suitable factors. This level of granularity constitutes our future work in online learning user preferences for personal activity scheduling.

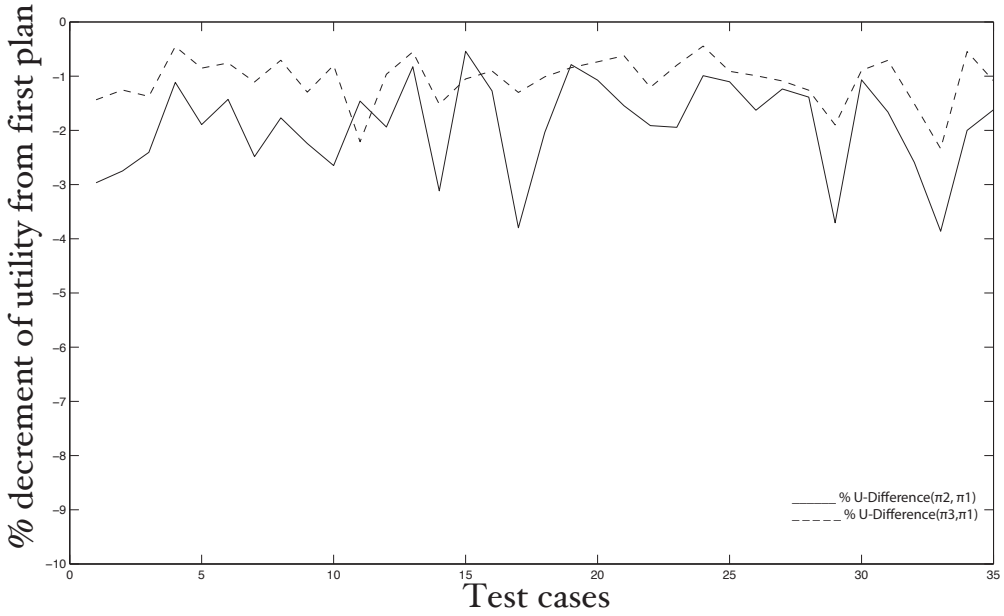

Figure 4-2: Percentage difference of utility between the alternative plans and the original

4.4 Evaluation

This section evaluates the computational methods presented in the chapter. Particularly, it starts with evaluating the multiple plan generation process and continues with the online learning process. The source code for the complete scheduler and the experiments, as well as the problem definitions and test results are available online.[1]

4.4.1 Generating Alternative Plans

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

We tested the multiple plans generation algorithm on 35 random test cases, ranging in size from problem instances of 6 activities to 36, in steps of 5 activities, with 5 problems for each size. We set the parameters used for the alternative plan generation as follows: $C = 1.0, W_D = W_L = W_{Time} = W_O = 0.25$. As the scheduler uses a stochastic process for post-optimization [4], we ran the above tests ten times for each problem and present the average values. From each run we obtained three plans. The learning process was disabled for these tests.

Figure 4-2 presents the average (over the 10 runs) utility change ratio for the two alternative

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

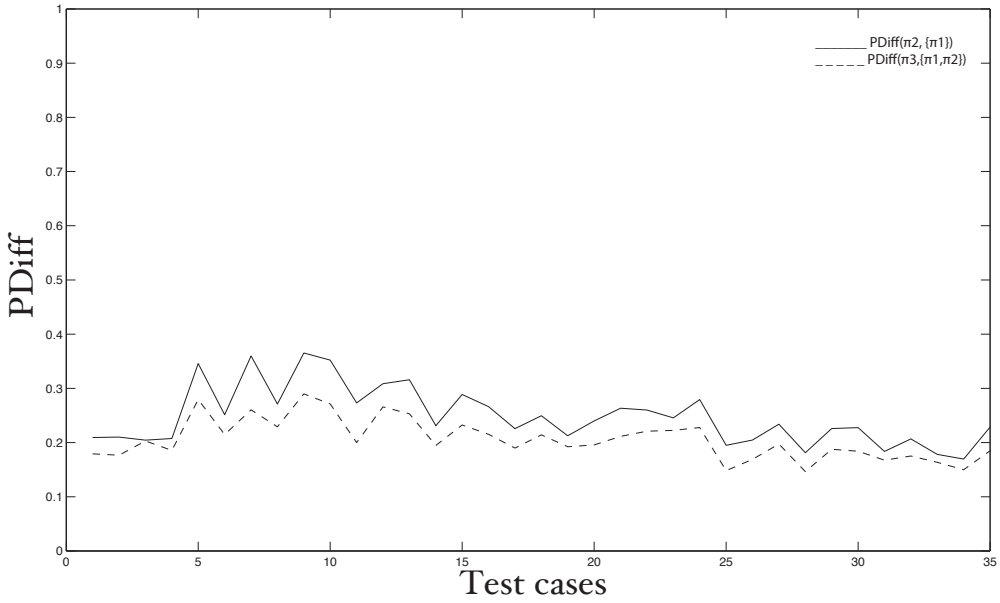

Figure 4-3: PDiff values for the two alternative plans for each test case

plans over the first one, for each one of the 35 problem instances of the test set. The solid line represents the utility change percentage for the first alternative compared to the original, whereas the dashed line represents the utility change percentage for the second alternative, compared to the original. To compare utilities, the pure utility function $U(\pi)$ (Formula 1.5) is used; $V(\pi)$ is used for the generation phase only.

As it can be seen from Figure 4-2, the two alternative plans are of slightly lower utility than the original one. The utility change percentage for the first alternative plan, compared to the original, ranges from -3.86% to -0.54%, whereas for the second alternative the utility change percentage ranges from -2.33% to -0.44%. The average utility change percentage for the first alternative plan is -1.9% and for the second alternative is -1.06%. As it can be seen, the second alternative plan has on average higher utility than the first alternative. This is because the second alternative is generated while trying to differentiate both from the original and from the first alternative plans. This makes it difficult for the second alternative to differentiate significantly from the original, while maintaining a satisfying utility. Thus, usually, the second alternative lies somewhere between the original plan and the first alternative, both in terms of differentiation, as well as in terms of utility.

Figure 4-3 presents the *PDiff* values, that is the distance, for the two alternative plans compared to the already found plans in each case. Particularly, the first alternative plan is compared to the original plan, whereas the second alternative plan is compared to the original plan and the first alternative simultaneously. As it can be seen, *PDiff* values are significantly higher for π_2 compared to π_1, than for π_3 compared simultaneously to π_1 and π_2. The minimum *PDiff*

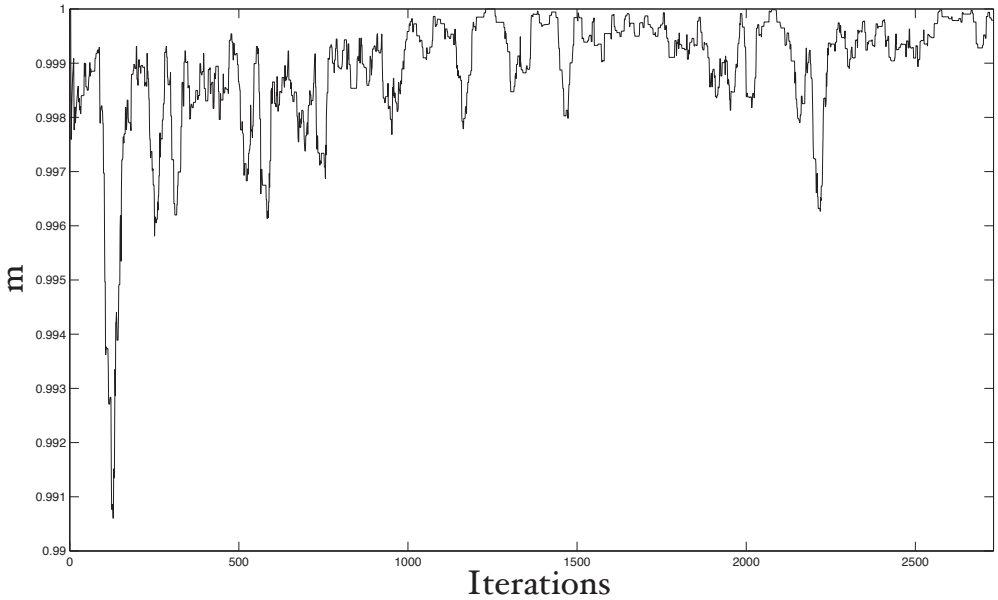

Figure 4-4: Learning convergence for the simulation of Virtual User 1 with $Q = 0.1$

value was 0.14 for the second alternative, whereas the maximum value was 0.36 for the first alternative. Another interesting observation is that *PDiff* values depend on the tightness of each problem, that is, the less activities within the same time period, the more chances for finding good alternative plans with significant differentiation. Indeed, in Figure 4-3 it is apparent a decrease in the *PDiff* values when moving to the right, i.e., towards problems with more activities.

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

4.4.2 Online Learning

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

Let $U'_k(\pi)$ be the modified version of $U_k(\pi)$, weighted by the k's virtual user preferences.

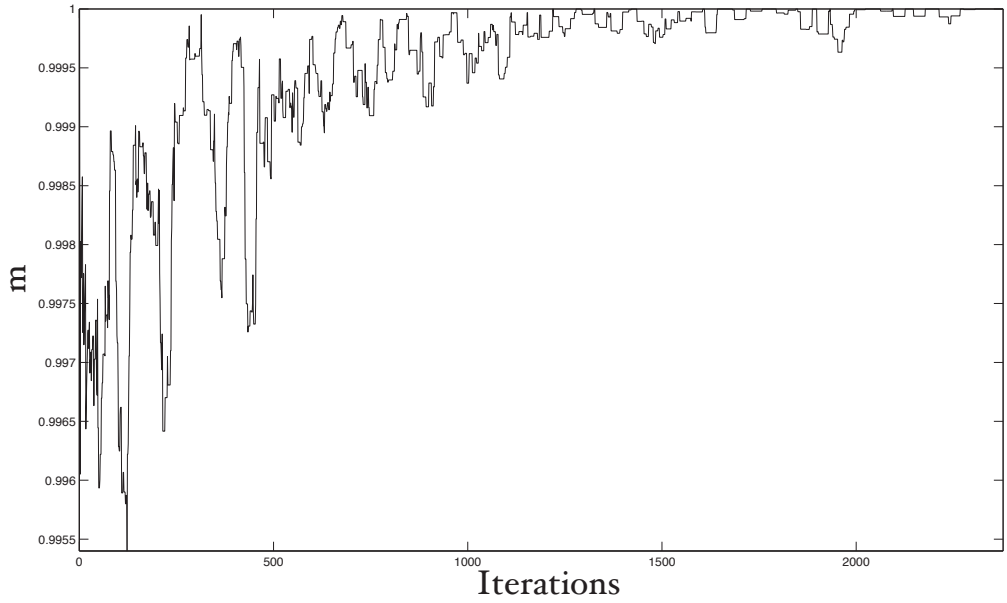

Figure 4-5: Learning convergence for the simulation of Virtual User 1 with $Q = 0.3$

These preferences are not known to the scheduler. So, the scheduler computes plans and attempts to guess the k's virtual user preferences using the online learning process presented in Section 4.3. For the k-th virtual user a random 30-activities problem is generated, the scheduler solves it and returns three plans to the user. These three plans are sorted in descending order according to the $U_k(\pi)$ values, using the estimated compensation factors, which are initially set to 1. Each time a set of three plans is returned to the k-th virtual user, the $U'_k(\pi)$ values of these plans are computed and the virtual user picks the one with the highest value, that is, according to its simulated preferences. If the plan picked by the k-th virtual user is not the same as the first plan returned by the scheduler, the compensation factors used by scheduler are adapted according to Formula (4.9). Then, a new random 30-activities problem is generated and the process is repeated until a termination condition occurs.

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

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

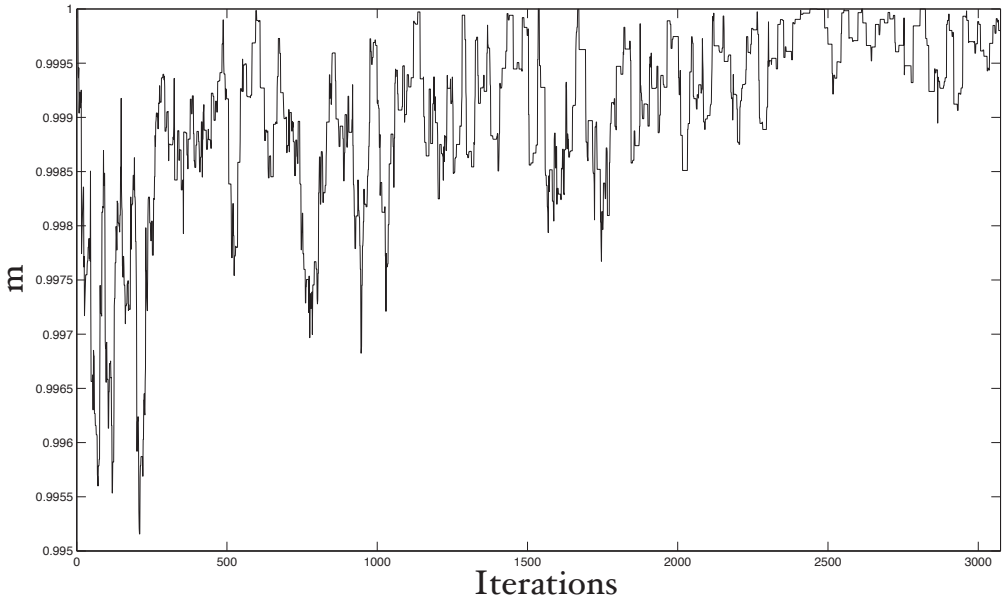

Figure 4-6: Learning convergence for the simulation of Virtual User 2 with $Q = 0.1$

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

$$m_i = \frac{\sum_{x=max(1,i-29)}^{i} \frac{U(\pi^x_{selected})}{U(\pi^x_1)}}{min(i, 30)} \tag{4.11}$$

where i is the problem instance being solved, $\pi^x_{selected}$ is the plan selected by the user for the $x-th$ problem instance, and π^x_1 is the first plan returned by the scheduler for the $x-th$ instance. This metric measures the relative quality of the (at most) last 30 plans selected by the user to the corresponding first plans suggested by the scheduler. If the user constantly selects the first plan suggested by the scheduler, then m_i tends to 1; on the contrary, if the user frequently selects plans of lower quality, m_i takes lower values. Thus, during online learning, we expect that m_i will start with lower values, which will increase towards 1 with the iterations.

Figure 4-4 presents the online learning method converging slowly to the first virtual user's preferences, with $Q = 0.1$, whereas Figure 4-5 demonstrates faster convergence for the same user and $Q = 0.3$. Similarly for the second virtual user, Figure 4-6 demonstrates slow convergence for $Q = 0.1$, whereas Figure 4-7 demonstrates faster convergence for $Q = 0.3$. All these figures demonstrate a behavior of the online learning method that is exactly what was expected, that is, the users select most frequently the first suggested plan, as online learning proceeds. Furthermore, the results demonstrate that better convergence, in terms of smooth-

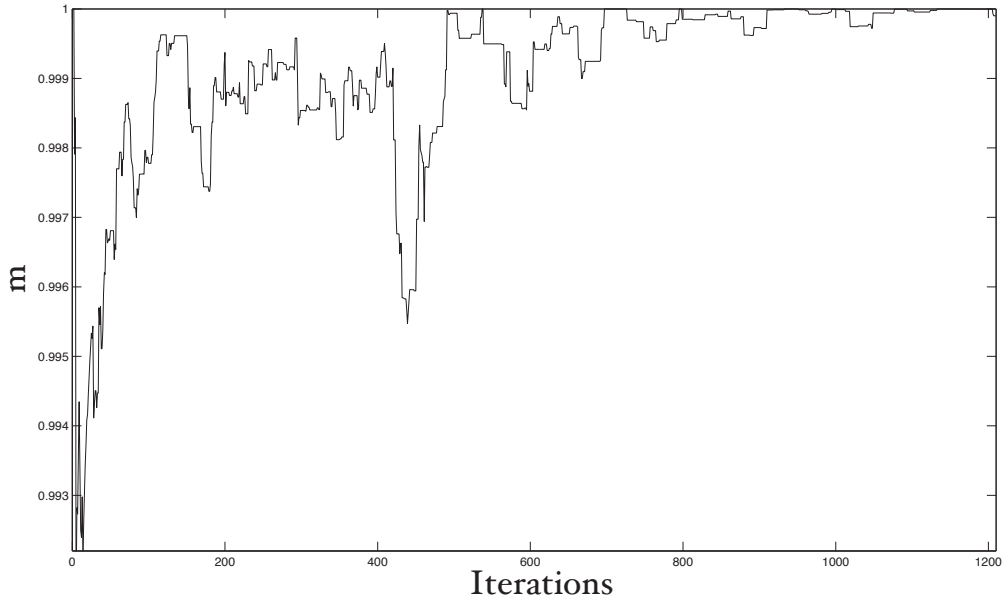

Figure 4-7: Learning convergence for the simulation of Virtual User 2 with $Q = 0.3$

ness and stability, is achieved with $Q = 0.3$ than with $Q = 0.1$. The observation that the users do not constantly select the first suggested plan, has to do with the fact that the real values of the compensation factors have not been learned yet with infinite precision.

4.5 Conclusions and Future Work

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

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

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

Future work includes extending the problem formulation with some new attributes, such as allowing the user to express his preferences over alternative locations, as well as employing activity ontologies so as to make it easier to express constraints and preferences over activities. The use of ontologies could also be exploited by the online learning mechanism, by allowing to reveal systematic errors performed by the user when evaluating alternative aspects of a plan, at the level of classes of activities. Concerning alternative plan generation, new metrics to measure the distance between two plans, either at the activity level or at the plan level, could be considered as well. Finally, we are actively working on combining all this work with meeting arrangements in a dynamic way, that is, using recursively the planning engine to reschedule already scheduled activities and meetings, in order to accommodate new meeting requests.

CHAPTER 5

Deployment and Evaluation of SelfPlanner

This chapter presents the initial version of the SelfPlanner electronic calendar prototype application. SelfPlanner uses the term *tasks* for activities. The initial version of the system was based on a simpler problem model than the one that was presented in Section 1.1.

5.1 Introduction

Traditionally, events and tasks are considered conceptually distinct aspects of a user's commitments and are treated separately by electronic personal assistants. Events usually have specific time and location reference and involve, explicitly or implicitly, other participants (e.g., a meeting, a class, an appointment with the doctor, etc.). On the other hand, tasks are mostly individual commitments having potentially a deadline to be accomplished (e.g., writing a paper or doing week's shopping); however, they are not given in advance a specific time window for execution, whereas location information is usually missing. This distinction is apparent in applications such as MS-Outlook[1] and, recently, in Google Calendar,[2] where events are put directly into the calendar, whereas tasks are kept in a separate task list. In this way, the user always has a clear view of what event is coming next or how overloaded her agenda is, but has not a clear view of what task should be performed next or how many tasks she has to accomplish within, for example, the next week. Although MS-Outlook supports drag-and-drop operations between the task list and the calendar, thus transforming a task to an event, people usually do not perform this type of manual scheduling but prefer to create short-term schedules on-the-fly, just by watching the task list or the reminders. In this way, they frequently break the deadlines of their tasks or do not accomplish them at all.

Other applications concentrate either on events or on tasks only. For example, Yahoo! Cal-

[1] http://www.microsoft.com/outlook/
[2] http://www.google.com/calendar/

endar[3] handles events only, whereas Chandler[4] focuses on collaborative tasks and information flow between users. None of these applications provides automated scheduling capabilities. A similar situation exists in research initiatives for managing personal time during the last two decades. As far as calendar management is concerned, research has concentrated for years on automating meeting scheduling, a widely agreed time-consuming process [121, 82, 11]. On the other hand, task management initiatives usually concentrate on bookkeeping the completion status of a task, exchanging messages between users that have been assigned the same task [30], and on raising reminders at appropriate times.

Our vision on managing individual tasks, that is, tasks where a single person is involved, considers them in a unified way with events, by assigning to them event attributes, such as temporal domain, duration, location, etc. In addition, we opt for a scheduling algorithm to reserve the necessary time for each task, to ensure its accomplishment, taking into account the user's stated preferences. In this way, the user can have a common view of her commitments, as well as a nearly optimized short- and near-term schedule. Furthermore, she can take more informed decisions about whether to undertake specific commitments (e.g., attending a university course with a well-defined workload over the semester) or not. Our vision has initially been expressed at Refanidis, McCluskey, and Dimopoulos (2004) [104].

This chapter presents SelfPlanner, a first step toward this vision. SelfPlanner is a Web-based application built on top of a fast domain-specific scheduler and integrates several Web technologies and Google-based applications. SelfPlanner treats tasks and events equivalently and constructs locally optimized schedules using a rich problem model. On the other hand, meetings involving other people can be arranged using manual procedures of Google Calendar (or third-party applications).

Apart from its scheduling engine, the two key features of SelfPlanner are the rich problem model and the innovative user interface. Both of them are critical in order for the system to be adopted by its potential target group, that is, people with very tight schedules (managers, academics, etc.). Concerning problem modeling, each task is characterized by a set of attributes such as duration, temporal domain, potential locations, interruptibility, etc. In case of periodic and interruptible tasks, additional information can be specified, such as the size of the period or the minimum and maximum duration of each part of an interruptible task. Ordering constraints between tasks, as well as several types of preference functions over the temporal domain of each task are supported.

Perhaps, the most tedious part of a task's definition concerns its temporal domain. This domain may consist of several temporal intervals distributed among large periods, for example, store working hours. In SelfPlanner, the user specifies temporal domains through a combination of template applications and manual editing. Templates are patterns that can be applied to large time periods. The user can create and reuse templates, as well as save her operations to reapply them to other tasks. Other user interface features concern a Google Maps-based

[3] `http://calendar.yahoo.com/`
[4] `http://chandlerproject.org/`

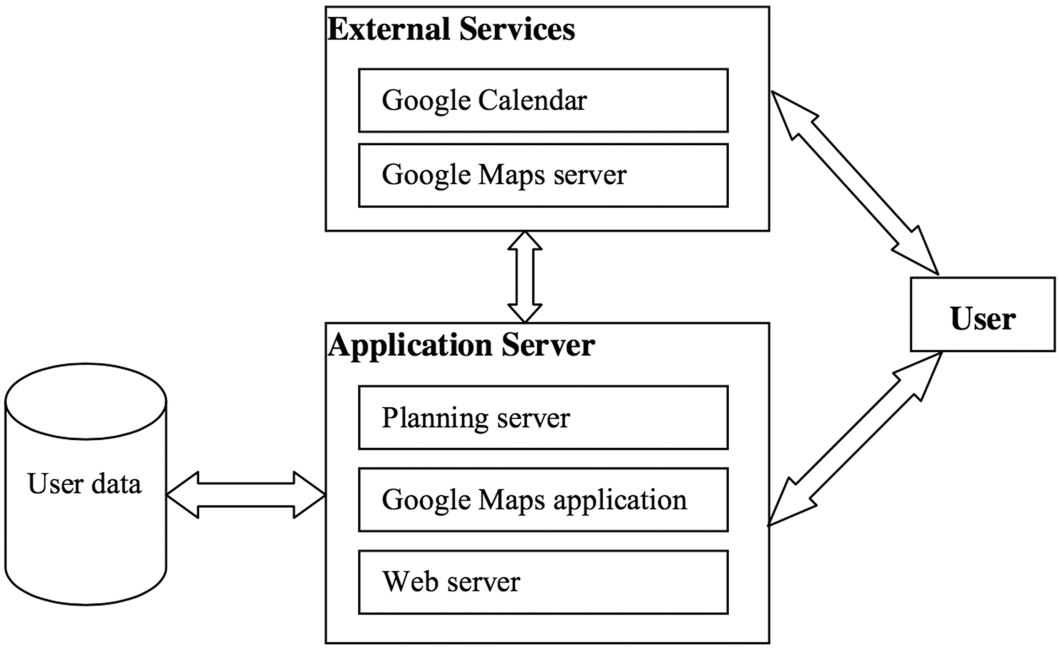

Figure 5-1: SelfPlanner overall architecture. The main system runs in the application server. External services are used by the system. The user can access her calendar both through the application server (using the dedicated SelfPlanner user interface) and directly in Google Calendar.

application that gives the user the option to define locations and compute temporal distances between them; classes of locations that provide alternative places where the user must be to accomplish a task; periodicity, where the instances of a task in different periods are scheduled separately; suggestions to relax constraints in case of overconstrained problems, etc.

The rest of the chapter is structured as follows: Section 5.2 presents the SelfPlanner system in detail, giving also illustrative use cases. Section 5.3 presents an extensive evaluation of the system, comprising an analytic and an exploratory part. Section 5.4 presents related work and, finally, Section 5.5 concludes the chapter and poses future directions.

5.2 The System

This section gives a detailed presentation of the implemented system, that is, its architecture, the key functionality points and some typical use cases.

5.2.1 Architecture

SelfPlanner is a Web-based application running in an application server, which comprises the planning server, the Google Maps application and the Web server (Figure 5-1). The planning server implements the SWO algorithm that is responsible for scheduling the user's tasks. It

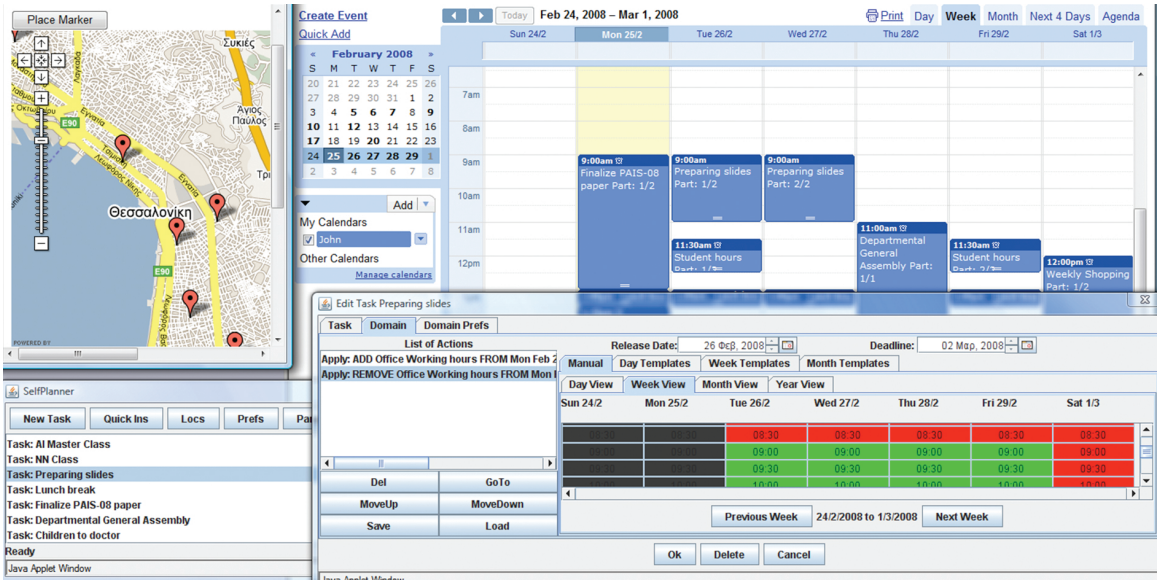

Figure 5-2: An overall view of SelfPlanner system. The main application window is shown at the lower left corner, now listing the active tasks. The "Edit task" dialogue box in the bottom right corner shows part of the temporal domain of the task "Prepare Slides." The Google Maps application for obtaining locations is shown in the upper left corner. Finally, in the upper right area, Google Calendar displays the user's current plan.

is also responsible to read the entries that are manually inserted into the user's Google calendars, as well as to update the user's Google calendars after scheduling her tasks. The Google Maps application manages the locations that are of the user's interest, computes their pairwise distances by querying the Google Maps server, and shares this information with the planning server. The Web server hosts the system's front end, which consists of a Java applet used for editing task details and HTML pages wrapping Google Calendar. Thus, the user interacts both with the Application Server, and with the External Services, that is, her Google calendars. All user data are stored centrally.

The application server encompasses several technologies and platforms. The overall system consists of a Java application (acting as a TCP/IP server), with the user interface being a Java applet (acting as a TCP/IP client). All connections between the user and the system are secure. The Google Maps application has been implemented in PHP and Javascript. The planning server consists of the Java application and the scheduler. The later that implements the SWO algorithm has been implemented in C++. Finally, user data, such as task details and the current plan, are retained as serializable Java objects (binary files).

5.2.2 Main Innovations

SelfPlanner builds on top of Google Calendar. The user can take advantage of all Google Calendar functionalities already familiar to her, as well as of the new functionalities added by SelfPlanner. This section concentrates on exactly these new functionalities, whereas it as-

sumes a basic familiarity with Google Calendar features. Figure 5-2 gives an overall view of the system, with the main windows open.[5] The main entity of the system is the "task." Using suitable user interface modules, the user can define all the parameters of a task, because they have been described in Section 1.1.

Perhaps, the most tedious job when defining a task is the definition of its temporal domain. SelfPlanner adopts an innovative way for defining domains, based on combining template applications and manual editing. A template is a pattern with a specific duration, for example, a day, a week, or a month, which characterizes time slots as available (*green*) or busy (*red*) time (see the "Edit Task" dialogue box in Figure 5-2). SelfPlanner provides a limited set of predefined templates; however, the user can create and store new ones. A template can be applied over the whole domain or over any part of it, in four different ways: adding the *green* slots, removing the *red* slots, removing the *green* slots, and adding the *red* slots. The user can apply several templates, whereas the order of application matters. On the other hand, manual editing allows the user to include or exclude (by clicking or dragging) specific time slots from a temporal domain (a minimum time slot of 30 minutes is assumed).

Another innovation concerns the way temporal domains are retained in memory: they are not retained as lists of temporal intervals but as lists of user actions. A user action is defined as either a template application or a manual addition/removal of a time slot in/from the domain. Thus, very large temporal domains consisting of numerous intervals can be represented very compactly through a small set of user actions. Efficient algorithms have been developed to answer questions such as whether a particular time slot is included or not in the temporal domain. The list of user actions is accessible through the user interface (see the left-hand side of the "Edit Task" dialogue box in Figure 5-2), which allows to modify it by changing the order of the actions or by deleting some of them. Any change in the list of actions is immediately reflected to the visual temporal domain representation.

SelfPlanner treats all tasks as interruptible, with non-interruptible tasks being characterized by $\mathtt{smin} = \mathtt{smax} = \mathtt{d}$. The decision in how many parts to split an interruptible task is taken by the scheduling algorithm. In addition, a task may be periodic. Periodic tasks are considered as collections of simple tasks. Each periodic task has a predetermined finite number of periods. The period may be either a day, or a week, or finally a month. The various instances of a periodic task are scheduled separately; thus, depending on the temporal domain of the task, they may be scheduled in different ways. For example, the first iteration of a weekly periodic task might be scheduled on Monday, whereas the second iteration might be scheduled on Thursday. In addition, the user may ask not to schedule an instance of the task for specific periods (e.g., holiday breaks). Periodic tasks may be interruptible as well.

A task is characterized by a set of possible locations, that is, to execute the task (or a part of it), the user must be in one of these locations. In case of interruptible and periodic tasks, different parts of the same task may be scheduled in different locations. To facilitate location

[5] A detailed description of the system, with all of its dialogue boxes, can be found in the online help, at `http://selfplanner.uom.gr`.

entry, SelfPlanner organizes locations into classes: a class is a set of distinct locations and it can be assigned to a task instead of a simple location. A location may be member of several classes. Traveling times are taken into account when scheduling tasks in different locations. Note that a special location, called anywhere, may also be assigned to a task. This location has zero distance from any other real location, meaning that the user can be anywhere to execute the task (e.g., making a phone call).

Ordering constraints are also supported by the system, with `before(A, B)` meaning that no part of task B can start its execution before all parts of task A have finished their execution. Ordering constraints are also supported for periodic tasks, provided that both tasks are periodic and with the same type of period. In that case, the ordering constraint applies to all pairs of instances of the periodic tasks that share the same period. For example, if A and B are weekly periodic tasks, with A having iterations over weeks 1, 2, and 3 and B having iterations over weeks 2, 3, 4, and 5, then the ordering constraint will apply to the pairs of instances that will be scheduled in weeks 2 and 3. In case of an ordering constraint between a non-periodic and a periodic task, the constraint applies between the non-periodic task and the first or the last iteration of the periodic one.

Concerning preferences over alternative schedules, the system supports unary preferences over the temporal domains of the tasks. In particular, the user can specify whether she prefers the task to be scheduled as early or as late as possible (linear descending and linear descending preference functions), before or after a specific time point (step preference functions) or whether she is indifferent on how a task will be scheduled within its temporal domain.

SelfPlanner also keeps track of the completion status of each task. This is not performed manually, having the system asking the user at the login time as well as at the scheduling time, whether already scheduled tasks or parts of them, whose scheduling time has passed, have been accomplished or not. Accomplished tasks are removed from the task list, whereas the durations of accomplished parts of tasks are subtracted from the remaining duration of their tasks. Tasks identified as non-accomplished are rescheduled.

In case of over-constrained problems, the system determines a maximizing subset of tasks that could be scheduled and updates the user's calendar with it. Maximization refers to the optimization quantity (1.5) (Section 1.1). The identification of this set is based on the best incomplete schedule that was encountered while trying to solve the scheduling problem using the SWO algorithm. Note that to achieve this functionality we changed the basic SWO cycle. In particular, each time a task or a part of task cannot be scheduled, the system attempts to schedule the remaining tasks using the same greedy construction algorithm. Then, the incomplete schedule is scored using Formula (1.5) only for the scheduled tasks or parts of them. If eventually no complete schedule is found, the best incomplete schedule is retrieved and presented to the user. Furthermore, the user is notified about those tasks that have not been scheduled entirely, being asked to relax the domains and/or the constraints involving these tasks.

Finally, it is worth to note that SelfPlanner supports time zones. Whenever the user edits a task, the temporal details of the task (release date, deadline, template applications, alternative

views of the domain, etc.) are related to the time zone of the user's computer, thus rendering the system internationally operable. This feature was critical to open the system for public use (Section 5.3.2).

5.2.3 Use Cases

This section presents six illustrative examples of tasks a user might want to insert in her calendar. The examples are ordered from the simplest to the more complex one, thus progressively taking advantage of more of the system's features. All use cases are supported by the current version of the system. However, this section serves an additional goal: to give the reader the seed to start thinking about potential new uses of intelligent calendar assistants technology and the changes they could bring in our life.

Case 1: Attend a Performance in the Theater. The performance will take place at the local theater, on Saturday evening, 9:00–11:00. This is the simplest type of task, since it is non-periodic, non-interruptible and is has a very precise time schedule and location reference. For such simple tasks, SelfPlanner offers a *Quick Insert functionality*, to facilitate data entry.

One might suggest that such simple tasks could be added directly into Google Calendar. This option is supported by SelfPlanner; however, it is not recommended because it does not allow the user to specify precisely the location reference of the task, which is important to compute traveling times. Furthermore, tasks directly added in Google Calendar reduce the flexibility in case of over-constrained problems, that is, the scheduling algorithm has no option not to include them in the user's schedule.

Case 2: Buying Gifts for the Children. This task employs two additional features of the system: the task has neither a specific time window to schedule, nor a specific location reference. Concerning time, the user should define the duration of the task, for example, 1 hour, a deadline, for example, before holidays, and the possible time slots when the task can be scheduled, for example, shopping hours, using a suitable template.

Concerning the location reference, there might exist several options in the area and the user might want to go to the nearest one with regard to other adjacently scheduled tasks. This is achieved by defining a class of locations, called, for example, Malls, that comprises all city's malls, and attach this class to the new task. While scheduling, SelfPlanner will select a specific location from this class to schedule the task, having as criterion to minimize traveling time with regard to adjacently scheduled tasks.

Case 3: Writing a Paper for a Conference. This is a typical example of a task with specific deadline. The novelty of this task is that it is interruptible. Apart from the deadline, the user estimates that she needs 30 hours in total to write the paper. She does not want to work for less than 2 hours and for more than 4 hours continuously, whereas her breaks should be at least

2 hours each.

Again a class of locations could be used to define plausible locations to write the paper, comprising, for example, home and office. The temporal domain of the task might be anytime excluding sleeping time, which can be defined using a suitable template. Finally, the user wants to write the paper as early as possible; thus, she attaches a linear descending preference model to this task.

Case 4: Teaching a Class. This is a simple task such as in Case 1, with the only difference that the task is weekly periodic. However, everything is well specified, that is, the time window is very precise within each week (e.g., every Thursday, 2:00–5:00), as well as the location reference (e.g., some University hall).

The procedure is similar to that described in Case 1, except that the user has to define the type of the period (weekly) and the number of repetitions (e.g., 13 weeks or provide a deadline). If the class will not take place at specific weeks, the user can easily remove these periods.

Case 5: Weekly Shopping. This is a combination of Cases 2 and 4. The task is non-interruptible, admits of scheduling, and is weekly periodic. To define its domain, the user has to employ a weekly template with the shopping hours. Furthermore, the user might prefer to perform weekly shopping as close to weekend as possible; thus, she can assign a linear ascending preference model to the task. Note that preference models for periodic tasks are applied separately to each instance of the task.

Case 6: Preparing for the Class. Before each class (Case 4), the instructor wants to devote 4 hours to recap the material and possibly revise it. This task can be performed in parts, with minimum duration for each part equal to 1 hour. Furthermore, revision should be completed before the class. To capture these interrelations, the user has to define this task as a weekly periodic one, with an ordering constraint between this and the Teaching a class one (Case 4). Thus, within each week, all parts of the preparing a class task should be scheduled before the Teaching a class task.

5.3 Evaluation

This section aims at evaluating various aspect of the SelfPlanner application. The evaluation comprises two parts: analytic and exploratory evaluation, with the latter concerning not only the current state of the system but also the general underlying idea of intelligent personal assistants.

5.3.1 Analytic Evaluation

We compared SelfPlanner to Google Calendar, a well-established calendar application. We selected Google Calendar for two reasons: first, it is also employed by SelfPlanner for presentation purposes and, second, it is a powerful representative of many commercial calendar applications. Note however that when we refer to our system, we do not take advantage of any of Google Calendar functionalities except from displaying the user's plan. Thus, the analytic evaluation concerns comparison on performing specific tasks using either the standard Google Calendar functionalities or the additional SelfPlanner ones that have been presented in the previous section.

For the comparison, we adopted the NGOMSL variation [66] of the GOMS model [25]. We based the comparison on four tasks that can be easily performed with and without SelfPlanner. While we measured the effort needed by the user for each task, we were not interested in the actual time (based on counting the number of keystrokes or mouse clicks, etc.) but in the number of simple subtasks needed to achieve each high-level goal. As a "simple subtask," we considered any user action such as "select a date," "open a dialogue box," "change the current view," or "apply a template." In this way, we estimated the cognitive effort required by the user to perform each task, something that is more tightly related to each specific approach, and not the actual time needed to perform each task, which depends on each specific implementation. Of course, defining what stands for a "simple subtask" depends on the user's experience; however, taking into account the expected target group of these applications and that any regular user may become experienced with them after some use, we decided to focus on above average desktop and calendar application users.

While designing the tasks, we had in mind that SelfPlanner differs in nature from traditional calendar applications in that it employs a scheduler. Thus, the tasks that we have designed require limited scheduling that, in case of Google Calendar, can be performed manually by the user (we did not take this cognitive effort into account); otherwise, the evaluation would not be fair. Especially for SelfPlanner, for the tasks where templates were used, we provide two alternative measures: one where the template(s) existed and one where the template(s) should be constructed from scratch using the system's default templates. Finally, we ignored locations references, because other systems do not support reasoning over them. A brief description of the tasks that were used in the analytic evaluation follows:

- *Task 1*: At Wednesday, February 4, 2009, 10 a.m. to 1 p.m., there is a Board meeting at your Department.
- *Task 2*: The spring semester of 2009 starts with the first week of February and ends with the last week of May. You teach an AI class each Tuesday, 2–5 p.m., excluding holidays (e.g., Easter).
- *Task 3*: At your university, you usually have a 30 minutes lunch break each weekday. The restaurant is open between 12:30 p.m. and 2:30 p.m. and you prefer to have the lunch as

late as possible. Schedule all your lunch breaks for the 2009 spring semester (excluding holidays), trying to avoid overlapping with other commitments (e.g., your AI classes).

- *Task 4*: You want to write a paper within the first 2 weeks of 2009 spring semester. You estimate that you need a total of 30 hours work for this task. You neither want to work less than 2 hours nor more than 4 hours continuously; your breaks should be at least 2 hours each. Your work has to be accomplished during your office hours.

Table 5.1: Comparison between SelfPlanner and Google Calendar Using the NGOMSL Model.

Task	SelfPlanner steps (without/with new templates)	Google Calendar steps
1	6	4
2	11	12
3	15/17	43
4	15/17	18

Table 5.1 gives an overview of the effort needed to perform these tasks, with and without the use of SelfPlanner.[6]

The four tasks used in our evaluation are typical representatives of the situations that might arise in practice while using calendar applications. Thus, Task 1 represents the simplest and most usual case, where a very specific event is added into a user's calendar. In that case, SelfPlanner is marginally less efficient than Google Calendar, something that was expected because Google Calendar is a highly optimized application in terms of user interface. Task 2 concerns the introduction of a constantly periodic task, a feature that is also supported by all calendar applications. Although again no scheduling is required, SelfPlanner takes advantage of its feature to allow removing specific periods at the time of task definition. On the other hand, Google Calendar requires that, after the task's definition, the user will move to the specific periods in the calendar and delete each instance separately.

Tasks 3 and 4 take advantage of the scheduling capabilities of SelfPlanner. Thus, Task 3 concerns a fluent periodic task, where all lunches are scheduled at 2 p.m. (as late as possible), except for Tuesdays where the lunches are scheduled automatically at 1:30 p.m. On the other hand, without SelfPlanner, the user has to go through all weeks and move the Tuesday lunches half an hour earlier manually. Finally, Task 4 concerns scheduling a non-periodic but interruptible task. In this case, SelfPlanner splits the task into parts and schedules it within a week, trying to avoid classes, meetings, and lunches. Without SelfPlanner, the user needs to split and schedule each part of the task manually. Figure 5-3 gives a snapshot of the user's calendar for the first week of the semester, after having accomplished all four tasks using SelfPlanner.

Note that the tasks used in the evaluation did not require rescheduling. However, if lunches were scheduled before classes (i.e., Task 3 was accomplished before Task 2), then rescheduling

[6] A detailed description of the actual steps for each system can be found at `http://selfplanner.uom.gr/FourSimpleTasks.html`.

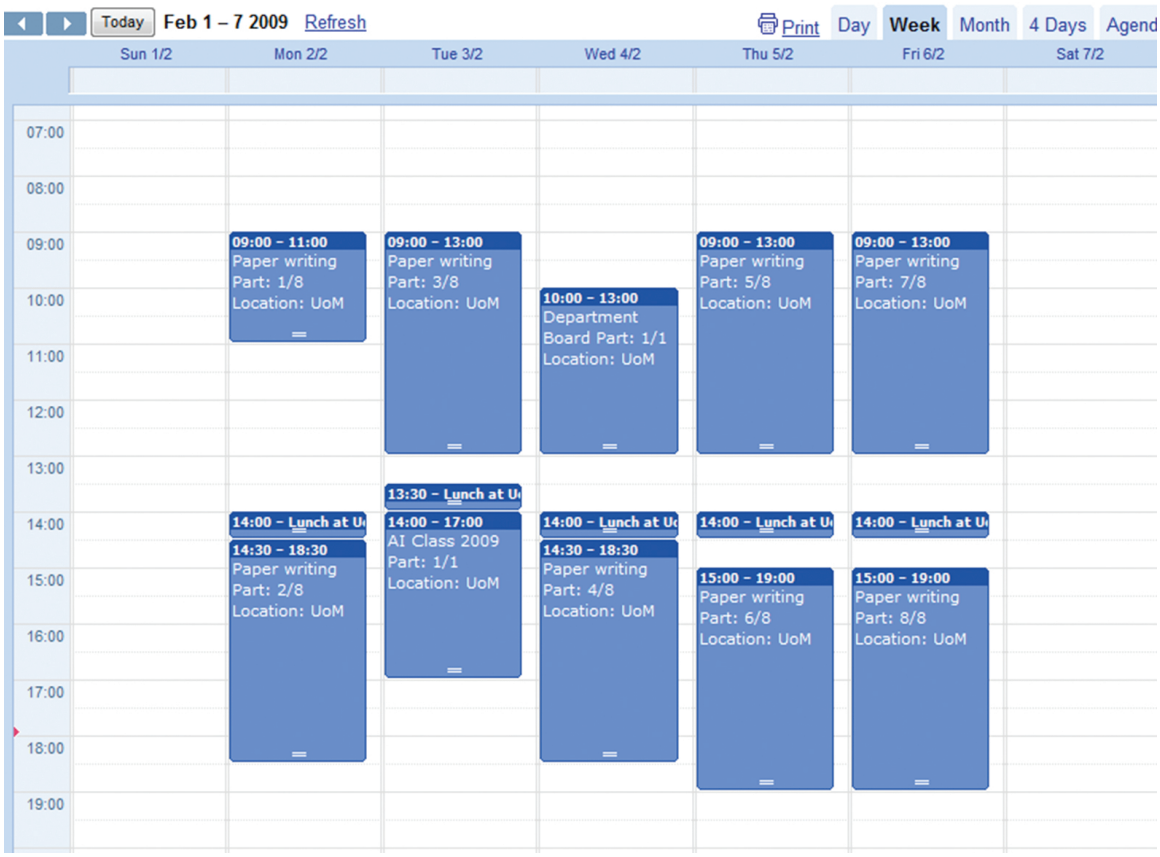

Figure 5-3: The user's plan for the first week of the 2009 spring semester, after accomplishing the four tasks of the analytic evaluation using the SelfPlanner system. Note the start time of the "Lunch" event on Tuesday with regard to the other days, as well as how "Paper writing" has been scheduled around the other events.

of the lunches would be required to schedule classes. In case of SelfPlanner, this would be done automatically; otherwise, manually.

The general result obtained from the analytic evaluation is that commercial calendar applications outperform SelfPlanner in terms of user's experience, when the introduction of a new event requires minimum scheduling. This was expected, because SelfPlanner is still a research prototype, where attention is mostly paid on functionality than on user interface efficiency. However, even in this case, the predominance of commercial calendar applications is marginal. On the other hand, the more scheduling or rescheduling is required when inserting a new task, the more evident the benefits from employing a scheduling engine become.

5.3.2 Exploratory Evaluation

SelfPlanner is an ongoing effort that has started in 2007 and continues till now. In October 2008, the system reached a mature state that made it possible to open it for the general public. Furthermore, online documentation has been released. Thus, this was a mature time for a first

exploratory evaluation.

Method

In October 2008, an open invitation to use the system has been sent to all major planning and scheduling mailing lists, resulting in more than 50 new registrations. In early December 2008, we asked all registered users to give us their feedback through a structured questionnaire.[7] We also gave them access to the four tasks used in the analytic evaluation; however, performing these tasks was not a requirement, because many users had used the system extensively, thus they had enough experience to complete the questionnaire.

Using questionnaires is a common practice in software engineering to evaluate interactive systems,with several prototypes such as QUIS® [27] or SUMI® [69] having proposed over the years. Our questionnaire is closer to the QUIS® one,with the user being asked to express her feeling over several statements using an integer scale from 1 to 5, with 1 meaning "Strongly disagree," 3 meaning "Indifferent," and 5 meaning "Strongly agree." An "N/A" (no answer) option was also provided.

The questionnaire comprised 40 statements, which are grouped in three parts (in, Nielsen and Mack 1994 [94], a number of questions between 20 and 40 are suggested for good questionnaires). The first part involved 25 statements concerning evaluating the system itself, such as *"Defining a new template for temporal domains was easy/intuitive."* This part gave us suggestions as to which parts of the system need further elaboration, in order for it to become more user friendly. The second part involved 10 statements concerning comparative evaluation between SelfPlanner and any other calendar application the user might have experience of, for example, *"SelfPlanner gave me a better overall view of my workload."* The third part involved five statements concerning the user's overall attitude toward the underlying SelfPlanner's idea, that is, using a scheduler to arrange individual tasks, and the potential extensions of it. An example of these statements is *"Provided that scheduling technology is mature enough, I would accept an intelligent personal assistant that knows all my tasks/events and proactively (re-) organizes them."* Finally, there was a fourth part, where users were asked to suggest the three most important extensions to SelfPlanner. They could select from a list of seven given potential extensions (e.g., "meeting scheduling" or "better scheduling engine") or write their own suggestions.

At the end of the evaluation period, we had received $n = 17$ questionnaires; however, for specific statements, the sample is smaller due to the "N/A" option. Due to the small number of questionnaires ($n < 30$), we used the well-known t-distribution and the independent one-sample t-test to verify the statistical significance of our hypotheses. Taking into account that the scale for the various statements is from 1 to 5, with $m_0 = 3$ corresponding to an "Indifferent" attitude, we used as null hypothesis the $H0 : m = m_0$, and we tested the hypotheses $H_1 : m > m_0$ and $H_2 : m < m_0$, where m is the mean value of the unknown real distribution. Statistical

[7] Both the system and the questionnaire used for the exploratory evaluation can be found at `http://selfplanner.uom.gr`.

significance coefficient was set to $\alpha = 10^{-3}$. The p-values for H_1 (or for H_2 when denoted) are given within parentheses.

All people were expert users of desktop applications and above average programmers; however, their prior expertise with calendar applications varied a lot. In the following paragraphs, we summarize the results, trying to focus on the most interesting of them.

Results

Concerning the first part of the questionnaire, people were satisfied by the Quick Insert functionality ($p < 10^{-6}$) as well as by scheduling new tasks using existing templates ($p < 10^{-4}$). However, they did not find it easy to define their own templates ($p \cong 0.02$) or to modify the action list of domain definition ($p \cong 0.06$). They also did not feel very confident with the whole temporal domain specification interface ($p \cong 0.01$), although they significantly liked the whole idea of using templates to define temporal domains ($p < 10^{-6}$)! These results can be interpreted as a suggestion for a more elaborated temporal domain definition interface (e.g., using wizards); however, to some extent, they can be ascribed to the limited time the users have worked with this innovative feature of SelfPlanner.

Preferences over the temporal domains were well accepted ($p \cong 10^{-5}$). People found intuitive the functionality for defining periodic tasks ($p \cong 10^{-9}$) and for omitting specific periods ($p < 10^{-5}$), with the latter being appreciated a lot ($p < 10^{-8}$). People found the interface for defining interruptible tasks intuitive ($p < 10^{-6}$), whereas they extremely liked the idea of supporting interruptible tasks ($p < 10^{-13}$). They did not find very intuitive the interface for defining ordering constraints ($p \cong 0.005$).

Defining locations and attaching them to tasks was considered intuitive ($p < 10^{-5}$ and $p < 10^{-8}$, respectively), whereas they liked a lot the idea of reasoning over temporal distances between locations ($p < 10^{-5}$). The interface for defining classes of locations was marginally considered intuitive ($p \cong 10^{-3}$); however, the general idea of organizing locations into classes was appreciated ($p \cong 10^{-4}$). Online documentation was also considered satisfactory ($p < 5 \cdot 10^{-4}$). Solving the scheduling problem and watching the resulting schedule on Google Calendar was considered easy ($p < 10^{-5}$), whereas the resulting schedules were as expected ($p \cong 5 \cdot 10^{-4}$). People liked a lot the interoperability with Google Calendar ($p < 10^{-9}$). The system was not considered very stable ($p \cong 0.007$), and the users would not use it in its current form ($p \cong 0.1$). However, the users liked the system very much and would use it in a more mature version ($p \cong 3 \cdot 10^{-4}$).

Concerning the second part of the questionnaire, this has been completed by 12 people only. The users compared mainly to Google Calendar; however, there were two references to MS-Outlook and one to Zimbra.[8] As expected, people had not considered SelfPlanner competitive to existing calendar applications when entering simple tasks ($p \cong 0,12$ for H_2) or simple periodic tasks ($p \cong 0.05$). However, they preferred SelfPlanner when omitting

[8] http://www.zimbra.com/

periods from periodic tasks ($p \cong 6 \cdot 10{-}4$), when scheduling fluent simple tasks ($p \cong 10^{-7}$), when scheduling fluent periodic tasks ($p < 10^{-5}$), when scheduling fluent interruptible tasks ($p < 10{-}4$), and when reasoning with locations ($p \cong 10^{-7}$). It has not been shown that SelfPlanner gave the users a better overall view of their workload ($p \cong 0.002$), whereas the resulting plans were not similar to what the user's would have created manually ($p \cong 0.39$). However, the latter statement is a bit ambiguous, because not being similar does not mean being worse necessarily; perhaps, this statement should be restated in a more precise way. Finally, and quite encouraging, the users claimed that they would consider to replace their existing calendar application with SelfPlanner, or they would like to see features of SelfPlanner being supported by their current calendar application ($p \cong 0.0002$).

As for the third, and perhaps the most interesting part of the questionnaire, the users strongly liked the idea of embedding a scheduler into calendar applications ($p < 10^{-10}$) and, provided that the scheduling technology is mature enough, they would accept an intelligent personal assistant to take control over their calendar and organize their tasks and events ($p \cong 6 \cdot 10^{-4}$). They do not consider it a problem using a calendar application that runs over the Internet, provided that the server is hosted by a trusted organization (or by the users themselves!) and the Internet connections are secured ($p < 10^{-4}$). Surprisingly enough, they are not very interested in using the system to automatically arrange meetings ($p \cong 0.003$), perhaps because they want to keep control over their calendar. Finally, they were somehow reluctant ($p \cong 0.11$) toward a system that knows their profile, collects information over the Internet and inserts tasks and events into their calendar proactively (e.g., attending a concert), and accomplishes tasks automatically whenever possible (e.g., buying tickets).

Finally, the three most suggested features for future extensions to the system were improvements to the current interface and the underlying scheduling engine, and support for overlapping events.

Discussion

The general conclusion we derive from the exploratory evaluation procedure is that SelfPlanner is a system with many innovative features that in many cases render it more efficient than existing calendar applications. On the other hand, it is also true that SelfPlanner is a research prototype and cannot be compared with commercial calendar applications in terms of maturity or attractiveness. However, it is very encouraging that users are positive to adopt it for managing their individual tasks either in its current form or in a more mature version, and we are working continuously toward this direction.

5.4 Related Work

SelfPlanner is the first system that attempts to schedule the individual tasks of a user's task list into her calendar using constraint propagation and optimization algorithms. Furthermore, it

introduces a new view of modeling this problem, including interruptible tasks, flexible periodic tasks, classes of locations, binary constraints, and preferences; such constructs do not appear in traditional calendar applications. In this section, we concentrate on research prototypes that offer some form of automation for calendar applications.

There are plenty of systems developed over the last 15 years that cope with the problem of automated meeting scheduling. Some of them concentrate on specific aspects of this problem [116, 43, 62, 117]. More recent efforts tend to incorporate learning components or to integrate with the Semantic Web. For example, RCal [121] is an intelligent meeting scheduling agent that assists humans in office environments to arrange meetings. The RCal agents negotiate with each other on behalf of their users and agree on a common meeting time that is acceptable to all users and abides by all the constraints set by all the attendees. The RCal supports parsing and reasoning about semantically annotated schedules over the Web, such as conference programs or recurring appointments [98].

The CMRadar [82] is an end-to-end agent for automated calendar management that automates meeting scheduling by providing a spectrum of capabilities ranging from natural language processing of incoming scheduling-related e-mails, to negotiate with other users or making autonomous scheduling decisions.

The PTIME [11] is an ongoing effort being developed under the CALO project [84], which aims at facilitating meeting scheduling. The innovation of PTIME lies at its capability to learn the user's preferences thus adapting its future behavior, whereas it emphasizes in adopting natural language for interfacing with the user. Part of the work in the PTIME thrust is Emma, a personalized calendar management assistant designed to help its user handle e-mail meeting requests, reserve venues, and schedule events [12]. Emma interfaces with commercial enterprise calendaring platforms and operates seamlessly with users who do not have Emma. It is designed to learn scheduling preferences, adapting to its user over time. The system is in initial deployment at SRI International.

Another effort within the CALO project concerns Towel [30], an initial attempt toward an intelligent to-do list. Towel allows the user to organize to-dos (group, tag, check, etc.) as well as delegate them to other users or agents. Although to-dos can be seen as tasks, Towel emphasizes on to-dos manipulation rather than in solving the scheduling problem associated with actually performing these to-dos. Furthermore, it does not support all the advanced modeling features of SelfPlanner.

5.5 Conclusions and Future Work

SelfPlanner is a deployed Web-based application targeting mainly expert users, especially those with heavy loaded agendas who are also willing to use e-calendars to organize their time. The system is constantly under development, with updated versions being uploaded several times a month. Its evolvement is driven both by user suggestions and by our overall vision. Of course, we do not overlook the fact that the system, although deployed, is still at an early stage,

with many things remaining to be done to become a truly intelligent electronic assistant.

As mentioned in the previous section, most of the effort to add intelligence to calendar applications concentrated on automating meeting scheduling. We could imagine two explanations for this: first, people think that meeting scheduling is the most difficult and time-consuming part of organizing a user's time. Although this might be true, meetings constitute a small part of our daily activities, whereas poor organization of the remaining tasks may result in significant waste of time and missed deadlines. Indeed, even scheduling together tasks with the same location reference, to avoid useless moves, would be of great value for many users.

On the other hand, we believe that the main reason why intelligent calendar systems do not focus, yet on automated scheduling of individual tasks is due to a hidden consensus that it would be very difficult for a user to accept a machine-generated daily plan of activities (e.g., for psychological reasons among others). Our experience from using the system is that this is not absolutely true: indeed, many of a user's tasks are very constrained, as for example giving a lecture or getting the children to school; thus, there is no need of scheduling for them. For the rest of the tasks, SelfPlanner gives the user; thus, many options to constrain the schedule and express her preferences that it is very unlikely to get an unacceptable plan. In any case, our experience has shown that through the actual interaction with the system people learn personal policies of how to use it and get the most from it.

As for the psychological concerns, we can witness our evidence: SelfPlanner has emerged, because a way to fulfill personal organization needs. The initial motivation was to develop an intelligent calendar system able to schedule together tasks with the same location reference. However, through the actual use of the first prototype, several other needs came up, with many of them being already implemented and described in this chapter. Interestingly enough, these claims are supported by our exploratory evaluation, where users said evidently that they like the idea of embedding a scheduler into calendar applications and, provided that the scheduling technology is mature enough, they would accept an intelligent personal assistant to take control over their calendar scheduler and organize their tasks and events! Furthermore, they do not have security warnings provided that the application is hosted by a trusted organization and secure connections are used.

As for the future, there are several directions in which SelfPlanner could be extended. We are taking very seriously into account the results of the exploratory evaluation and work toward making the user interface more efficient and intuitive (especially the part that concerns the temporal domain definition). We also work on the underlying scheduling engine, to produce even better plans. Furthermore, our immediate future plans involve support for task priorities, overlapping events, multiple user calendars, editing of the resulting plan, as well as the development of a mobile interface for the client part of the system. Integration of meeting scheduling capabilities is also under consideration.

However, the most challenging extension concerns its transformation to a planning system. We imagine this next-generation system as possessing semantic knowledge about the user's current state, a rich ontology with actions having preconditions and effects, and a set of goals

to be achieved. The system will help the user to insert tasks into her calendar to achieve goals or subgoals. For example, the user might set a goal for attending a conference, which could generate a sequence of actions being inserted into her calendar, including preparing the slides, bookings for the trip, and traveling. Classical planning algorithms for causal reasoning can be used to solve the planning problem, probably in a mixed-initiative manner.

CHAPTER 6

The Current Version of SelfPlanner

SelfPlanner was extended, to provide two of the three most requested improvements that were mentioned in the previous chapter. These were (a) improvements to the current interface, (b) improvements to the underlying scheduling engine and (c) support for overlapping events. We improved the underlying scheduling engine by replacing it with the one presented in Chapter 3 and we provided support for overlapping events by implementing the newer problem model presented in Section 1.1. We also added the enhancements presented in Chapter 4, that is multiple plan generation and on-line user preference learning.

There were other improvements as well. SelfPlanner used a simpler version of the problem model that did not support many utility sources. By porting it to the current model, plus the model extension with support for non-monotonic temporal preferences that we presented in Chapter 3, we did not only provide support for overlapping activities but also for richer user preferences. It is now possible, for example, to specify a preference for the period of the day that an activity should be scheduled, i.e., in the morning, afternoon or evening. Moreover, since SelfPlanner originally appeared there has been at least one other system following our approach [9].

In this chapter we will take an in-depth look at the internals of SelfPlanner 2.7, the current version, as well as the SelfPlanner API's Object Oriented model. We will also present an overview of how SelfPlanner was utilized in the myVisitPlanner project. Moreover, we will present another extension to the problem model that provides traveling time minimization. Finally, we propose an extension to the system for the handling of joint activities.

6.1 SelfPlanner Protocol

SelfPlanner is based on a client-server model, where the server consists of a Java application and the scheduler, which was described in the previous chapters. The Java application acts as a TCP/IP server and listens for connections on TCP port `7021`. A client opening a connection to the server socket will receive SelfPlanner's greeting along with its version number and Self-

Planner's public key.[1] The client will have to generate an AES session key and transmit it to the server encrypted with the use of the server's public key. From this point on, all communications will be assumed from the server to be encrypted with the generated session key. To log-in the client will either have to request to log in using the `AUTH_LOGIN` message and then provide a valid username and password (this is the way the official SelfPlanner client connects) or request to log in using the `MVP` message (this is the way a third-party program using the API can connect). In the second case, the server will generate a nonce and send it to the client. The client will have to respond by transmitting it back to the server encrypted with the third-party program's authorized private key.[2]

The client can then send a problem instance definition in SelfPlanner's Objected-Oriented model[3] using the `SEND_SPDATA` message and then ask for a solution using either a `SOLVE_REQUEST` or a `MULTI_SOLVE_REQUEST` (for multiple plan generation) message. All the data transmitted and received are serializable Java objects from the Objected-Oriented SelfPlanner model. The full SelfPlanner protocol with all the messages the server sends and recognizes is included in Figure 6-1.

6.2 Periodic Activities

As was described in the previous chapter, SelfPlanner extends the problem model by directly supporting periodic activities. These are activities that are repeated either daily, weekly or monthly. When SelfPlanner generates a problem instance for the scheduler[4] the number of periods of each periodic activity are calculated and the application inserts one activity per period in the problem definition that has a temporal domain inside that period only (which is extracted from the periodic activity's temporal domain). All binary constraints and preferences involving two periodic activities or a periodic activity and a non-periodic one are expanded to constraints and preferences involving the periods of the activity.

6.3 Real World Locations

Real world locations can either be specified in the official client by using Google Maps (where their traveling distance will be queried from Google) or they can be specified in the API using their latitude and longitude information. In the second case the programmer of the third-party application using the API will have a choice among the following options: (a) providing the traveling time himself, (b) querying Google's Distance Matrix service for the traveling time between the locations or (c) estimating the distance. The API includes the `LocationUtils`

[1] SelfPlanner stores its RSA key-pair in the `keys.sp` file. If the file is missing a new key-pair will automatically be generated when the server starts.

[2] The nonce provides protection against replay attacks. See "Setting up the API" in Appendix A.

[3] For an example see Appendix B.

[4] See Appendix D for an example of a problem instance definition that the scheduler recognizes.

```
public interface MyPlannerProtocol {
    int MY_PLANNER_PORT = 7021;
    String AUTH_LOGIN = "Login";
    String AUTH_SIGNUP = "Sign-up";
    String AUTH_SUCESS = "Auth succeeded";
    String MVP = "MyVisitPlanner-Coordinator";
    String MVPSP = "MyVisitPlanner-Coordinator-Joined";
    String REQ_PASS = "Password:";
    String SEND_LOCAL_TIMEZONE = "Sending Local Timezone data";
    String SEND_TASKS = "Sending Task data";
    String SEND_SOLUTION = "Sending Solution data";
    String SEND_BCONSTRAINTS = "Sending Binary Constraint data";
    String SEND_BPREFERENCES = "Sending Binary Preference data";
    String SEND_DMINCONSTRAINTS = "Sending Distance min Constraint data";
    String SEND_DMINPREFERENCES = "Sending Distance min Preference data";
    String SEND_DMAXCONSTRAINTS = "Sending Distance max Constraint data";
    String SEND_DMAXPREFERENCES = "Sending Distance max Preference data";
    String SEND_IMPCONSTRAINTS = "Sending Implication Constraint data";
    String SEND_IMPPREFERENCES = "Sending Implication Preference data";
    String SEND_DOMAINDATA = "Sending Domain data";
    String SEND_TASKMANAGER = "Sending Task manager";
    String SEND_LOCATIONMANAGER = "Sending Location manager";
    String SEND_LOCATIONCLASSMANAGER = "Sending Location Class manager";
    String SEND_BINARYMANAGER = "Sending Binary manager";
    String SEND_DISTMINMANAGER = "Sending Distance min manager";
    String SEND_DISTMAXMANAGER = "Sending Distance max manager";
    String SEND_IMPMANAGER = "Sending Implication manager";
    String SEND_SPDATA = "Sending Self Planner Data";
    String SEND_TEMPLATEDATA = "Sending Template data";
    String SEND_TEMPLATEMANAGER = "Sending Template manager";
    String SEND_DOMAINMANAGER = "Sending Domain manager";
    String SEND_LOCATIONCLASSES = "Sending Location Class data";
    String SEND_PROPERTIES = "Sending Properties";
    String SEND_SWO_PARAMETERS = "Sending SWO Parameters";
    String USE_SP_ACC = "Use SP Account";
    String USE_USERS_ACC = "Use User's Account";
    String SEND_WEIGHTS = "Sending Learning Weights";
    String REMOVE_BCONSTRAINT  = "Remove Binary Constraint";
    String REMOVE_BPREFERENCE = "Remove Binary Preference";
    String REMOVE_DMINCONSTRAINT  = "Remove Distance min Constraint";
    String REMOVE_DMINPREFERENCE = "Remove Distance min Preference";
    String REMOVE_DMAXCONSTRAINT  = "Remove Distance max Constraint";
    String REMOVE_DMAXPREFERENCE = "Remove Distance max Preference";
    String REMOVE_IMPCONSTRAINT  = "Remove Implication Constraint";
    String REMOVE_IMPPREFERENCE = "Remove Implication Preference";
    String REMOVE_LOCATION = "Remove Location";
    String REMOVE_TASK = "Remove Task";
    String REMOVE_TEMPLATE = "Remove Template";
    String REMOVE_DOMAIN = "Remove Domain";
    String REMOVE_LOCATIONCLASS = "Remove Location Class";
    String REMOVE_GROUP = "Remove Task Group";
    String REMOVE_COMPLETED_TASKS = "Remove Completed Tasks";
    String REMOVE_EXPIRED_TASKS = "Remove Expired Tasks";
    String REMOVE_BOTH = "Remove Both Completed and Expired Tasks";
    String RELOAD_TEMPLATES = "Reload Templates";
    String LOAD_LOCATIONS = "Load Locations";
    String LOAD_CALENDARS = "Load Calendars";
    String UPDATE_GOOGLE = "Update Google Calendar";
    String SOLVE_REQUEST = "Solve System";
    String MULTI_SOLVE_REQUEST = "Multi Solve System";
    String MULTI_SOLVE_AT_ONCE_REQUEST = "Multi Solve At Once System";
    String SOLVE_SUCESS = "System Solved";
    String SOLVE_PARTIAL = "System partially solved";
    String SOLVE_FAIL = "Could Not Solve System";
    String VALIDATE_OK = "Validated";
    String VALIDATE_FAIL = "Could not validate";
    String VALIDATE_REQUEST = "Validate Subtasks";
    String VALIDATE = "Subtasks validated";
    String JVALIDATE_REQUEST = "Just Validate Subtasks";
    String JVALIDATE = "Just Subtasks validated";
    String GOOGLE_SUCCESS = "Updated Google Calendar";
    String GOOGLE_FAIL = "Could Not connect to Google account";
    String GOOGLE_FAIL2 = "Could not connect to Google account while updating";
    String RETRIEVE_WEIGHTS = "Retrieve Learning Weights";
    String CHANGE_PASSWORD = "Change password";
    String SET_GOOGLEACCOUNT = "Set Google Account";
    String AUTH_TO_GOOGLE = "Auth To Google";
    String DISCONNECT = "!Quit";
    String OK_DATA = "OK";
    String OK_SUP = "Sign-up OK";
    String OK_SUP_AUTH = "Sign-up needs AUTH";
    String ERR_DATA = "Corrupted data";
    String ERR_NOSUP = "Not supported";
    String ERR_NOLOGIN = "Unrecognized Login";
    String ERR_WRONGPASS = "Wrong Password";
    String ERR_TOO_MANY_CLIENTS = "Too many clients connected";
}
```

Figure 6-1: SelfPlanner Protocol

class which provides methods for querying Google for the traveling distance or estimating it automatically. Distance estimation is done by providing the average traveling speed and by using the following formula to estimate the distance in kilometers:

$$2C_{ER}\sin^{-1}\sqrt{\cos(C_{DR}lat_a)\cos(C_{DR}lat_b)\sin^2(\frac{1}{2}C_{DR}(lng_a - lng_b)) + \sin^2(\frac{1}{2}C_{DR}(lat_a - lat_b))} \tag{6.1}$$

where C_{ER} is a constant representing Earth's radius in meters, C_{DR} a constant specifying the degrees to radius conversion, lat_a the first location's latitude, lat_b the second location's latitude, lng_a the first location's longitude and lng_b is the second location's longitude.

6.4 Memory of Past Activities

The scheduler has no memory of past activities. SelfPlanner on the other hand can keep an account of which activity parts have been completed. For those activities it will decrease their d_i^{min} and d_i^{max} values by the sum of the durations of the parts completed the next time it will generate a problem instance for the scheduler.

In addition, for providing the scheduler with the current location information, so as that location distances from the current location are taken into account, the following method is used: SelfPlanner will insert an activity at the $[-1, 0)$ interval that has a high utility and has to be scheduled at the current location. This location is ignored when SelfPlanner parses the scheduler's solution and its utility is subtracted from the value of the objective function (1.5). Because the scheduler supports intervals starting from 0, all domain intervals' a and b values are incremented by 1 and the inserted activity is scheduled at $[0, 1)$. When the solution is parsed and the extra activity is ignored, every t_{ij} is decremented by 1.

6.5 Domain Representation

From a theoretical point of view, the temporal domain of an activity consists of a set of intervals. For practical reasons we consider only integer domains. In the context of the application, a unit corresponds to the quantum of time that, as in most electronic calendars, is 30 minutes. For reasons of clarity, in the following we will use a notation of the form ‹DD/MM/YY HH : MM›to denote time points. Depending on the context, several parts of the time stamp will be omitted or altered.

Using intervals to represent domains is however problematic both from a computational and from a user's experience point of view. Suppose for example that a user wants to schedule an activity of 2 hours duration and this activity has to be performed during office hours next

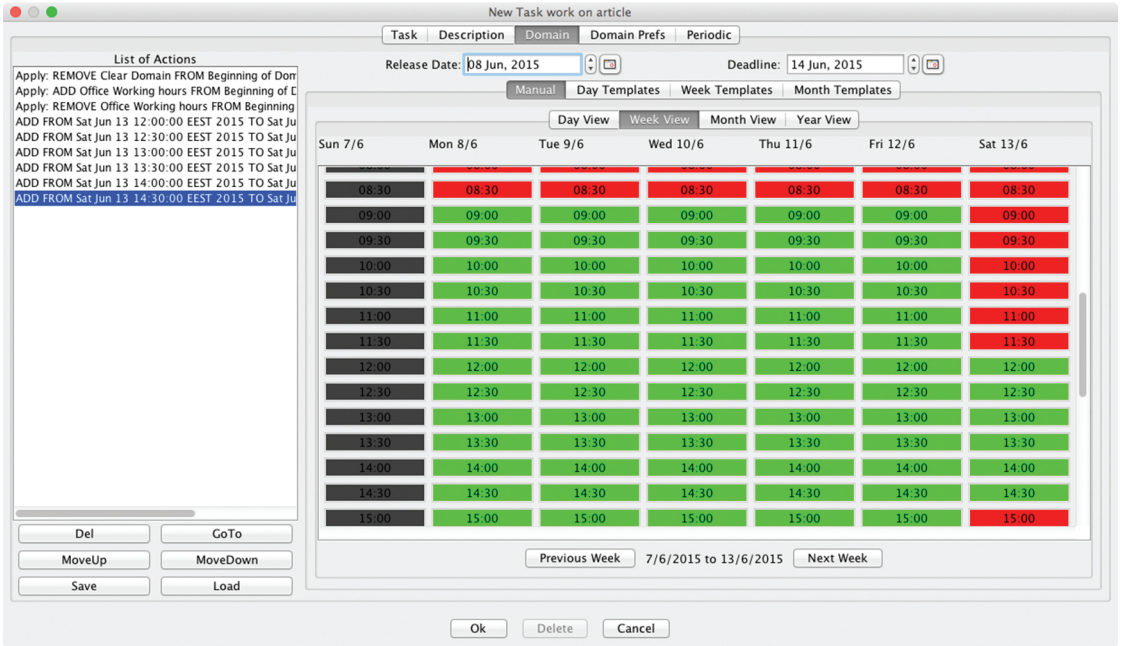

Figure 6-2: Editing the temporal domain of an activity in SelfPlanner

week. Supposing a 5 day working week, this might results in five intervals of the form, say:

$$[\langle 27/10/14\ \ 09:00\rangle, \langle 27/10/14\ \ 17:00\rangle) \ldots [\langle 31/10/14\ \ 09:00\rangle, \langle 31/10/14\ \ 17:00\rangle)$$

Imagine now what happens if the same activity has a deadline after a month or a year: Storing and retrieving the domain of this activity would be a time- and space-consuming process. Even worse, having the user to define this domain would be an inhibitory factor to use the system at all. To overcome these deficiencies, we selected to avoid using interval representation for temporal domains.

Templates and List of Actions

A template is a pattern with specific duration and with no absolute time reference. SelfPlanner supports three types of templates: Daily, Weekly and Monthly. Each template consists of a set of intervals covering the entire pattern's period and denoting which slots are allowed for an activity's execution (the remaining are not). For example, a daily template of lunch hours would comprise a single interval, say $[\langle 12:30\rangle, \langle 15:00\rangle)$. Similarly, a weekly template of office hours would comprise five intervals of the form $[\langle \mathrm{Mo}09:00\rangle, \langle \mathrm{Mo}17:00\rangle)$ to $[\langle \mathrm{Fri}09:00\rangle, \langle \mathrm{Fri}17:00\rangle)$. Note that the daily template's interval does not have any day reference, whereas the weekly template's intervals have a relative reference of the week's day.

Templates can be used to define domains. The simplest way is to combine a template with a release date and a deadline. However, to increase flexibility in domain definition through templates, we distinguish four different ways in applying a template. These are the following:

- Add included, denoted with ⇧: The time slots identified by the template are added in an activity's temporal domain.
- Remove excluded, denoted with ⟮: The time slots not identified by the template are removed from an activity's temporal domain.
- Add excluded, denoted with ⟯: The time slots not identified by the template are added in an activity's temporal domain.
- Remove included, denoted with ⇩: The time slots identified by the template are removed from an activity's temporal domain.

As an example, consider again the daily lunch hours template, named *Lunch*. If we want to include lunch hours to an activity's domain we use ⇧*Lunch*. If we want to say that an activity is to be executed only during the lunch hours, we use ⇧⟮*Lunch*. If we want to exclude lunch hours from an activity's domain we use ⇩*Lunch*.

List of Actions and Temporal Domains

A temporal domain can be defined through a sequence of template applications. For example, one activity's domain might consist of office working hours excluding the lunch hours. This could be defined by a list of actions, that initially adds office hours to the temporal domain and then removes lunch hours.

More formally, a *domain action* is defined as a temporally constrained template application. This is denoted by providing an absolute interval with the template, e.g., ⇧⟮*Lunch*@[‹27/10/14 00 : 00›, ‹28/10/14 00 : 00›). Domain actions that are not temporally constrained apply to the whole activity's domain.

A *list of domain actions* is an ordered sequence of them. For example, the following list defines office working hours excluding the lunch hours (note that domain actions are not temporally constrained, so they apply to the whole activity's domain):

⇧⟮*OfficeHours*,
⇩*Lunch*

Manual editing a domain, i.e., adding or removing a time slot without the use of a template, can be seen as a special case of template application. Suppose we have a daily template named *All*, consisting of the single interval [‹00 : 00›, ‹24 : 00›), i.e., the whole day (operators ⟮ and ⟯ are meaningless for this template). So, manually adding (removing) an interval to (from) an activity's temporal domain is equivalent to applying ⇧*All* (⇩*All*) temporally constrained over this interval.

Finally, a *temporal domain* is defined as a list of domain actions, accompanied by a release date and deadline. So, the following specifies an activity's domain over the week from 27/10/14 to 31/10/2008, including all office hours but the lunch hours:

Algorithm 1. *GetTimeSlotStatus*

Inputs: A domain represented by a list of domain actions and a time slot T.
Output: Either of the INCLUDED or EXCLUDED values.

1. If T is before the release date or after the deadline, return EXCLUDED.
2. Let D be the last domain action. If no such action exists, then D is NULL.
3. While D $\neq$ NULL
   If D adds the T, return INCLUDED.
   If D removes T, return EXCLUDED.
   Let D be the previous domain action. If no such action exists, D is NULL.
4. Return INCLUDED.

Figure 6-3: **Algorithm 1.** GetTimeSlotStatus

[‹27/10/14 00 : 00›, ‹31/10/14 24 : 00›)
⇧➿*OfficeHours*,
⇩*Lunch*

The semantics of a domain are the following:

1. All time slots before the release date or after the deadline are excluded from the domain.
2. A time slot is included in the domain, if there is a domain action that adds this time slot in the domain, whereas no subsequent domain action removes the time slot.
3. A time slot is excluded from the domain, if there is a domain action that removes this time slot, whereas no subsequent domain action adds the time slot.
4. All unspecified time slots are considered as included in the domain.

6.5.1 Computational Issues

Using lists of domain actions to represent temporal domains gives rise to interesting computational problems, such as whether a time slot is included in the domain or not, how to transform the domain into the traditional representation with list of intervals or, finally, how to simplify the list of domain actions. These issues are treated in the following subsections.

Domain Inclusion

Knowing whether a time slot is included in an activity's temporal domain or not is important, among others, when graphically displaying parts of the domain on the screen. The algorithm

Algorithm 2. *GetIntervals*

Inputs: A domain represented by a list of domain actions.
Output: A list of intervals.

1. Let A be a table of integers, whose size equals the number of time slots be-tween the domain's release date and the deadline. Let S be this size. Initialize A with zeroes. Let $C = 0$.
2. Let D be the last domain action. If no such action exists, then D is NULL.
3. While $D \neq$ NULL and $C < S$.
   For each time slot T added by D
   If $A[T]$ is 0, then set $A[T]$ to 1 and increase C by 1.
   For each time slot T removed by D
   If $A[T]$ is 0, then set $A[T]$ to -1 and increase C by 1.
   Let D be the previous domain action. If no such action exists, D is NULL.
4. If $C < S$
   For each time slot T such that $A[T] = 0$, set $A[T] = 1$.
5. Create a list of intervals by joining consecutive time slots having $A[T] \geqslant 0$.

Figure 6-4: **Algorithm 2.** GetIntervals

presented in Figure 6-3 answers this question.

Algorithm *GetTimeSlotStatus* is very fast. Indeed, suppose that the action list has N entries and each template has at most M intervals, then the worst case complexity is $O(N \cdot M)$, with N and M usually taking small values.

List of Intervals

It is often required to transform a temporal domain represented by a list of domain actions to the traditional list of intervals. For example, most existing schedulers do not support the list of domain actions representation. So, algorithm *GetIntervals* is a generalization of algorithm *GetTimeSlotStatus* and does exactly that. It is presented in Figure 6-4.

Algorithm 2 is very fast, however it might have significant memory requirements in case of large domains due to the definition of the temporary variable A. However, an alternative design that would directly encode the new domain in intervals would be inefficient, since step 3 would require to traverse the entire list of intervals in order to decide whether a time slot has already got a status or not.

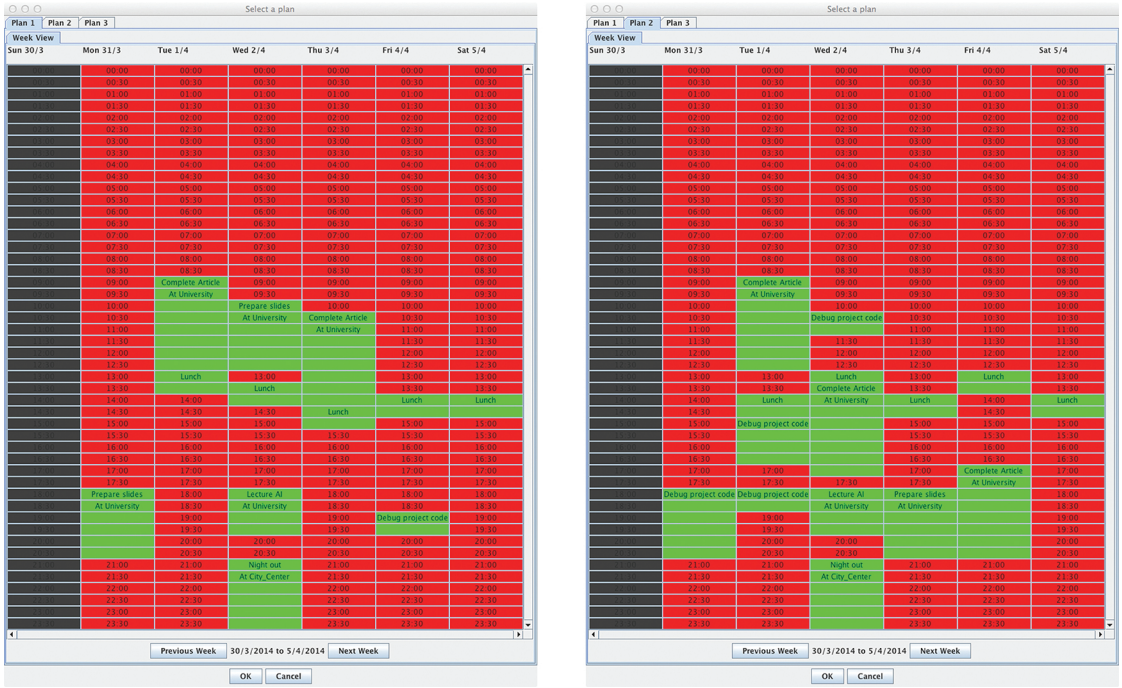

Figure 6-5: Selecting a plan in SelfPlanner

Simplifying Domains

There are cases where several domain actions can be removed from the list without any change in the resulting domain. For example, suppose ⇧🕐*OfficeHours* exists in a domain action list. This domain action adds to the domain all the included template's intervals and, at the same time, it removes all excluded template's intervals. Furthermore, this domain action is not temporally constrained, so it covers the entire domain. In this case, any domain action occurring before this one would not have any effect in the domain and thus it could be safely removed.

Detecting domain actions that can be safely removed from the action list requires a simple change in Algorithm 2. In particular, in step 3 we should check, for each domain action D, whether the domain action has increased C or not. In the latter case the domain action does not affect the temporal domain and can be removed. However, from an application point of view, this removal should (and does) not occur without prior confirmation from the user, since the user might intend to remove or modify some of the subsequent domain actions, which could result in ineffective domain actions to become effective.

6.6 Multiple Plan Support in SelfPlanner

We extended SelfPlanner to support the generation of alternative plans, as well as the online learning mechanism. Each time the user requests a new plan, a number of alternative plans are presented to him, using the current values of the compensation factors, as it has been described in the previous sections. The plans are sorted in descending order according to their weighted

$U(\pi)$ values. When the user chooses his preferred plan, SelfPlanner adapts the compensation factors accordingly.

Figure 6-5 gives an overview of the new GUI features of SelfPlanner. Three alternative plans are presented in different tabs, with the main plan (Plan 1) being by default selected. For each plan, the user can see how his activities have been scheduled (green cells), as well as his free time (red cells). By pressing the "ok" button, his choice is recorded and the compensation factors are changed accordingly.

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

6.7 Plan Generation Process

In Figure 6-6 we present a flow chart of SelfPlanner's plan generation process. Having specified her activities and constraints or preferences over them, the user can issue a solve request. SelfPlanner will first read the user-indicated Google calendars and get any custom events, i.e., non-SelfPlanner events that have been added directly in Google Calendar. These events are considered as busy time and their temporal interval is transparently removed from the domains of all SelfPlanner activities during the solving process. Next, the current problem instance is exported to the scheduler format, which is based on the problem model described in Section 1.1, and the scheduler is called. The scheduler will produce a new plan, using the SWO algorithm and will subsequently post-optimize it using Simulated Annealing (SA).

The scheduling process will be repeated until the number of alternative plans requested by the user has been reached, while trying to maximize the deviation between those plans by using the methods presented in Section 4.2. Subsequently, the user will have to choose a plan among them. This will result in the weights used for learning the user's preferences being adapted by using the on-line learning method of Section 4.3. Finally, the system deletes all not-yet-started SelfPlanner generated entries from her Google Calendar, and updates her calendars with the selected plan (each activity is posted in a potentially different user-specified calendar).

6.8 Object-Oriented Model

A user's problem instance is represented as an instance of the class `MyPlannerData`. An instance of this class holds references to the user's learning weights, to the u_L constant (see Section

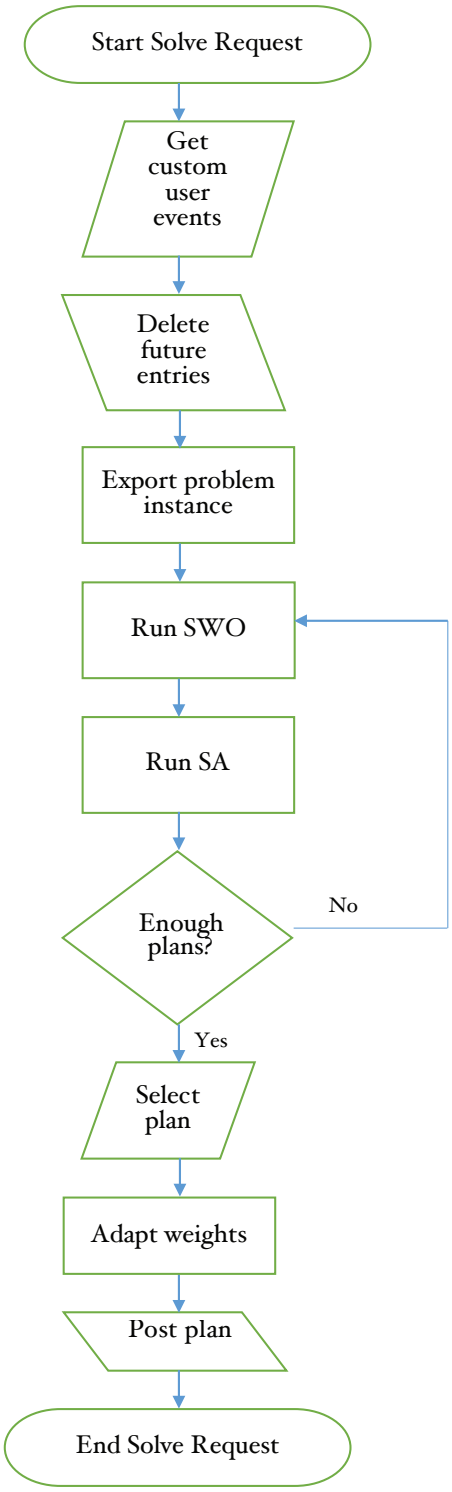

Figure 6-6: SelfPlanner's plan generation process

6.10), to the user's settings and to a collection of *Managers*, which are instances of collection classes that hold the various aspects of the user's problem instance, along with utility methods for managing them. The *Manager* classes are:

- `TaskManager` which collects the user's `Task` objects. A `Task` object can be periodic/non-periodic and interruptible/non-interruptible. Each `Task` holds a `Domain` object which defines the temporal domain of the activity using `ManualAction` objects and `Template` objects (which are subclasses of the `DomainAction` class).
- `LocationManager` which collects the user's `Location` objects. Each `Location` object represents a physical location, along with the distances between these locations, which are represented as `LocationPair` objects.
- `LocationClassManager` which collects the user's `LocationClass` objects. Each `Location-Class` is a set of `Location` objects specifying the set of possible locations where an activity's parts can be scheduled.
- Four instances of the `BinaryManager` class. Each instance refers to a specific type of binary constraints and preferences (i.e., ordering, proximity, implication). In addition, each instance collects `BinaryConstraint` and `BinaryPreference` objects.
- `TemplateManager` which is a collection of pre-defined `Template` objects.
- `DomainManager` which is a collection of pre-defined `Domain` objects.

The UML diagrams of these classes are included in Appendix A.

6.9 myVisitPlanner

myVisitPlanner [103] is a web application focusing 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 a typical visit duration is, where the museum is, etc. Furthermore, myVisitPlanner supports an ontology of activities, whereas the user can express her preferences over this ontology (as well as for particular activities). Thus, by selecting a cultural activity, myVisitPlanner is able to schedule the activity within the user's calendar, while taking into account her preferences. The system's welcome screen is shown in Figure 6-7.

myVisitPlanner uses SelfPlanner as the system's planning engine. myVisitPlanner's planning module, the *Trip Planner* (which is written in Java) defines the user's scheduling problem using the SelfPlanner classes presented in the previous section. Afterwards, it connects to SelfPlanner using the API to obtain three high-quality valid schedules which it stores in the system's MySQL database. The plans are read from the front-end and displayed to the user.

Some innovations of the system, in regards to scheduling individual activities, and how *Trip Planner* handles them are described:

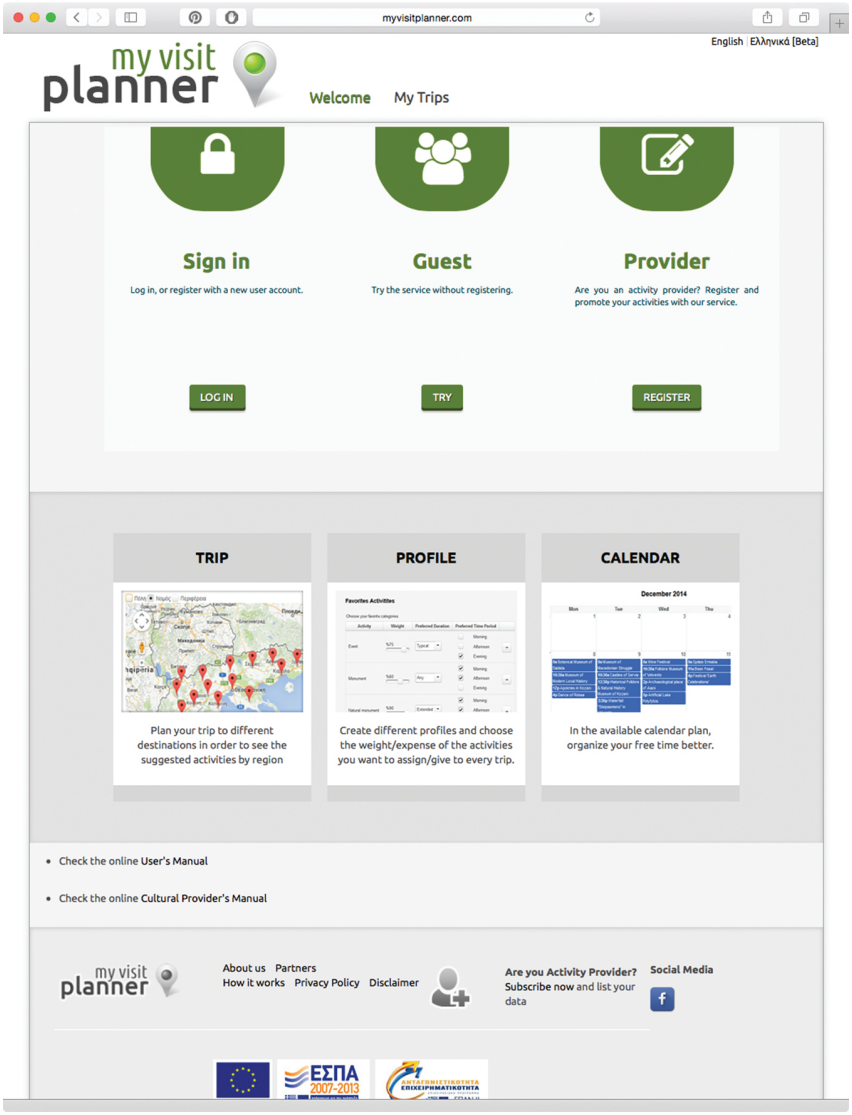

Figure 6-7: myVisitPlanner welcome screen

Managing the User's Sleep Hours

The system automatically inserts a daily periodic activity that represents the user's sleep hours. This activity has a domain between 23:00–10:00, a minimum duration of five hours and a maximum duration of eight hours. The activity's utility is high so that it will always be inserted into the schedule, whereas the extra duration utility is high enough so that it will be scheduled at the maximum possible duration only when doing so will not force the scheduler not to insert another activity in the plan. The activity's location is the user's resting place (e.g., hotel) so that the traveling distance from both the last evening activity and to the first morning activity are taken into account. These activities do not appear in the plan shown to the user.

Dispersed and Condensed Plans

A user can define a preference for a dispersed or a condensed plan. To produce these plans `BinaryPreference` objects are defined, of either the binary minimum proximity type (for dispersed plans) or the binary maximum proximity type (for condensed plans), between every pair of activities that either maximize or minimize the temporal distance between the scheduled activities over the trip's temporal domain.

Starting and Ending Locations

The myVisitPlanner model allows activities that have different starting and ending locations. For example a "bus tour" activity can be specified where the user will be at a different location at the end of the activity. To model different starting and ending locations we define `Location` objects per combinations of starting and ending locations that appear in the selected activities. When computing the traveling distances between locations, and by taking advantage that the distance matrix Dist can be asymmetric, we compute distances *from* the combined `Location` object by using its ending location and we compute distances *to* it by using its starting location.

Activity Bundles

It is possible for a cultural provider to provide groups of activities, called *bundles* that have to be included in the user's plan together. To model these we use `BinaryConstraint` objects of the binary implication type so that all activities in a bundle have to be placed in the user's plan or not be placed at all.

6.10 Traveling Time Minimization

One important omission in the problem model (Section 1.1) is that the objective function does not minimize the traveling time between different locations. It will only minimize them *indirectly* if the problem instance is over-constrained. In that case, the scheduler will have to minimize the distances to produce high-utility plans that schedule all the activities included at their maximum possible durations.

We added the following objective to Formula (1.5) by defining a utility for the total time *not* spend traveling in a plan π_i:

$$U^{travel}(\pi_i) = u_L \times \frac{width_D - \mathrm{TTime}_{\pi_i}}{width_D} \tag{6.2}$$

where u_L is a constant and $width_D$ is defined as:

$$width_D = \arg\max_m b_{m,F_m} - \arg\min_n a_{n,1} \tag{6.3}$$

and the time spend traveling $\mathtt{TTime}_{\pi_i}$ in plan π_i:

$$\mathtt{TTime}_{\pi_i} = \sum_{x=\arg\min_n a_{n,1}}^{\arg\max_m b_{m,F_m}} \begin{cases} \mathtt{Dist}(l_{ij}, l_{kl}) \text{ if } \exists t_{ij}, t_{kl}, t_{ij} = x, t_{kl} > t_{ij} \wedge \nexists t_{yz}, t_{ij} < t_{yz} < t_{kl} \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{6.4}$$

Traveling Time Minimization is disabled by default, as it was defined and added in a later time in the scheduler than the research described in the previous chapters. It was added as a requirement of the myVisitPlanner project. It can be enabled by defining the constant u_L using the `-j` parameter. In the SelfPlanner API it can also be defined by setting the u_L constant through the `setLocationDistanceUtility` method of class `MyPlannerData`.

6.11 Joint Activity Scheduling

SelfPlanner is capable of planning the user's time by handling her individual activities. However, an important group of activities cannot be handled by the current system. These are *joint activities*, that is activities involving other people as well (e.g., meetings). To handle this type of activities the system needs to be extended with a negotiation protocol that allows users of the system to define joint activities together. In this section we propose such a protocol.

Two types of protocols can be defined, one requiring a Central Authority (CA), such as a central SelfPlanner server that all users connect to, and a peer-to-peer protocol where each user runs her own SelfPlanner system copy. We propose a joint activity protocol for each of these approaches. In all cases there should be a degree of *fairness* for managing conflicting user preferences, which could be handled with the use of voting methods [5]. Both of these protocols are based on the meeting negotiation protocol defined by Sen et al (1997) [118].

Sen et al's protocol is defined as follows:

1. The host (meeting initiator) attempts to find some time intervals that suit her. She sends these intervals to the invitees.

2. Each invitee receives these intervals and tries to find local solutions that satisfy the constraints of the meeting. He then sends their proposals (as bids) back to the host. The proposals can be subsets of those sent by the host, or they can be counter-proposals.

3. The host collects the replies from the invitees and evaluates their bids. If they all (including the host) include a common interval, the meeting is scheduled for that inter- val and awards are sent back to the invitees. If no common interval is found, the host generates a new proposal, consisting of a set of time intervals, based on the bids of the invitees and her own calendar. She sends that proposal back to them.

4. Upon receiving the reply, the invitees reply as in step (2). If an award is received, for their bids, they mark the available time slot in their calendars. Otherwise they reject the

proposed slots.

To apply the above protocol to SelfPlanner's domain model (Section 6.5) we define the *Combined Domain* of a joint activity as the set of that activity's temporal domains as defined by each participant. The two domain algorithms presented in Section 6.5 are modified so as to check if a time slot is included in each participant's temporal domain and instead of returning included or excluded for a time slot, they return (for a particular user):

- included: If the time slot is included in all the participants' domains for the joint activity.
- excluded: If the time slot is not included in any of the participants' domains for the joint activity.
- some_def: If the time slot is included in the user's and at least one of the other participant's domain.
- others_def: If the time slot is included in at least one of the other participant's domain but not in the current user's.
- all_others_def: If the time slot is included in all other participants' domains but not in the current user's.
- user_only_def: If the time slot is included only in the current user's domain.

In the SelfPlanner GUI we use the color green for included time slots and the color red for excluded time slots. These values can be color-coded as well so a user defining her version of the temporal domain of a joint activity with others would immediately notice the other participants' temporal domains. Another option would be to represent others_def using a range of values from $\frac{1}{n}$ to $\frac{n-1}{n}$, where n is the number of participants, and color code these using different shades of a single color, so that the current user will immediately be able to tell how popular is the choice of a time slot by the other participants. The same approach could be used to represent some_def using a range of values as well and using different shades of another color to color code them. A reference *Combined Domain* implementation is included in Appendix A.

6.11.1 Central Authority Protocol

To define a joint activity, the activity initiator (called the *host*) will have to create it in SelfPlanner, set all its field and the user-ids of the other participants (called the *invitees*). Subsequently, the following protocol is initiated:

1. The host creates the joint activity and sets her version of the temporal domain of the activity.

a. All the invitees receive the host's joint activity proposal, along with the host's domain for that activity. Each user will create a *Combined Domain* involving the host's domain proposal and her own version.

b. When all the participants define their domains, the system will remove those intervals belonging to other scheduled activities or joint activities and will attempt to find a common interval.

c. If there is more than one common interval, the system will solve each user's problem instance,[5] for each common interval. For example, if there are two common intervals $[20, 23)$ and $[60, 63)$, each user will solve two problem instances, where the first instance places the joint activity at the first interval, whereas the second instance places it at the second interval. If the number of common intervals are larger than a constant C_{MI}, C_{MI} problem instances will be defined and solved. The interval that maximizes the sum of all users' objective functions will be selected for the joint activity. If there is no common interval, we go to the second round.

2. All the users' domains are expanded automatically so as to include previously scheduled activities of the users (but not previously scheduled joint activities).

   a. Step 1.c is repeated automatically with the modification that user activities in the common intervals are rescheduled. If there is no common interval we go to the third round.

3. All users' *Combined Domains* are refreshed to the full set of the participants' domains and they are notified that the joint activity could not be scheduled with the current preferences and/or without rescheduling other joint activities. The users can alter their domains again if they wish, while viewing the full *Combined Domain* of the joint activity. Their new domains are stored.

   a. When all users modify or confirm their domains Step 2.a is repeated automatically. If there is no common interval we go to the fourth round.

4. All the users' domains are expanded automatically so as to include previously scheduled joint activities of the users.

   a. Step 1.c is repeated automatically with the modification that both user activities and joint activities in the common intervals are rescheduled. If there is no common interval the participants are notified and given a choice between canceling the joint activity or repeating step 3 with the modification that the users' previously scheduled joint activities intervals are available (if there is a common interval in their scheduled time).

   b. If a common interval is found that falls during a user's previously scheduled joint activity, that user is notified and the protocol is initiated by that user to that joint activity's

[5] At each problem instance, all previously scheduled activities of each user have their domains set at their scheduled intervals and locations so that they will not be moved.

participants to reschedule it.

Rounds 2 and 4 are automatically conducted by the system and do not require the involvement of the participants, whereas rounds 1 and 3 require all the participants' involvement.

6.11.2 Peer-to-Peer Protocol

The protocol can be modified to run in a peer-to-peer fashion, without the requirement of a Central Authority. Each participant will have to run her own SelfPlanner server and have the user-id, IP address and public key of the host's SelfPlanner copy, while the host will need to know the user-ids, IP addresses and public keys of all the participants.

The following modifications have to be made to the protocol:

- In round 1 the host signs her proposal and domain with her SelfPlanner copy's private key and connects to each participant's SelfPlanner copy. The authenticity of the request can be verified as the participants have to know the host's public key. The participants (except the host) do not need to know the others' public keys; the host can send them along with the joint activity proposal signed with her private key, along with their IP addresses and user-ids.

- In round 1, when the participants define their own domains they send them directly to each other participant of the joint activity, signed with their private keys. Each domain received by each participant is verified first with the use of the sender's public key before refreshing the receiver's *Combined Domain*. The domains are pruned automatically by the senders' SelfPlanner copies to omit intervals belonging to scheduled activities or joint activities before being broadcast to the other participants.

- When each participant receives all the others' signed domains she will attempt to find the common intervals. Subsequently each participant will run step 1.c on her SelfPlanner copy, but with the modification that they maximize their own utility.

- When a common interval is found, each user will broadcast to the others the best n common intervals (if more than one) that maximize their total utility, along with their utility values. If the intervals broadcast do not match, the joint activity interval will be selected via an automatic voting process where all participants' utilities for each common interval are taken into account.[6]

- In round 3 the users will broadcast new domains to the other users (or the confirmation of their previous ones). This time though each user's SelfPlanner copy will not prune the other scheduled activities' intervals from the joint activity domain.

[6]There are various ways to model this step. For example, each user could select the interval that maximizes the sum of all the participants utilities (like the CA protocol does) and retransmit that interval to all the others. If the transmitted intervals do not match, the common interval is selected via an automatic majority voting process.

- If round 3 is repeated, the SelfPlanner copies will not prune the other scheduled joint activities' intervals from the domain broadcast.

Whereas the peer-to-peer protocol allows the host to send the credentials of the other participants to each invitee, it would be better if each invitee has the other participants' credentials.[7] Otherwise, a lot of trust is placed on the host which she could exploit to deceive the users by sending them false credentials. Each user's SelfPlanner copy will compare the credentials of the other participants as sent by the host with her own versions and warn the user if they do not match.

The involvement of each user in the peer-to-peer protocol is the same to the CA one if the host is trusted. If she is not, each user will have to obtain the other participants' credentials and compare them with the ones transmitted by the host.

[7] That is their user-ids, their SelfPlanner copy's IP address and public key.

CHAPTER 7

Conclusions

In this thesis we explored the problem of planning individual activities, with the goal of defining the electronic calendar of the future: an *intelligent* calendar where the user will not only record her events and view them back, but have the calendar itself plan her activities and produce her schedule, while taking into account all her preferences and the traveling time between the locations of her activities. We selected the Constraint Optimization Problem formulated in Refanidis and Smith (2010) [105] as the underlying computational model of an intelligent calendar and extended it with non-monotonic temporal preferences, that provide the user with a more flexible preference model for the scheduling time of her activities, and with traveling time minimization that was missing in the original problem.

We modeled the above problem using Constraint Logic Programming and defined and tested various heuristics to obtain an optimal solution to instances of the problem. The problem proved to be too complicated to solve real world instances of it using this approach and we subsequently explored metaheuristic methods for solving it.

We defined and implemented a modified Simulated Annealing algorithm, empowered with Tabu Lists and backtracking, and coupled with in-domain heuristics that is comparable to SWO (the state of the art algorithm presented for the above problem) when it is given the same execution time. Furthermore, the algorithm can produce better solutions when it is given more execution time whereas SWO cannot improve further upon them. However, the best strategy for hard problem instances involving a large number of activities is to produce an initial solution using SWO and have the modified SA post-optimize it.

In addition, we extended the problem model by quantifying the difference between plans and utilized these definitions to extend the scheduler so as to produce multiple plans for each problem instance. An on-line learning method for the user's preferences was also presented that will converge the main plan presented to the user to the user's qualitative preferences given enough use of the scheduler.

Moreover, we presented SelfPlanner—our vision of the electronic calendar of the future—that utilizes this research to plan the user's time. SelfPlanner can be used as stand-alone web

application that utilizes Google Calendar for the presentation of its plans and Google Maps to specify real world locations or it can be used by third-party calendar applications or programs through the use of an Object-Oriented API. SelfPlanner was selected and used as the planning engine of the myVisitPlanner project.

Finally, we proposed an extension and presented two versions of a negotiating protocol that will allow the system to handle joint activities. By implementing one of the two protocols, SelfPlanner will be able to handle meeting scheduling as well.

7.1 Future Directions

The research presented in this thesis can be extended in a number of directions. Concerning the Object-Oriented model used by the full SelfPlanner system it could be extended either with the use of ontologies and/or by applying HTN planning techniques so as to define complex activity categories that consist of simpler activity categories that are partially ordered. The underlying Constraint Optimization Problem model is rich enough as the set of the low-level activities can be defined in it and connected with the use of binary constraints and preferences. The scheduler can be used on the expanded problem instance without alteration. If new limitations are identified in the problem model, while defining the higher-level model, the problem formulation could be extended again, as it was done in this thesis, and the extensions can be implemented in the scheduler.

Concerning joint activity scheduling, the two proposed protocols should be implemented and evaluated in real world usage scenarios between different parties. These tests should involve both honest users using the system with the goal of reaching an agreed time interval and malicious users trying to exploit the protocol for their benefit.

Concerning the criticism of the users at the evaluation of the original SelfPlanner prototype, the GUI of the system was not improved to address their concerns. The system's user-interface should be replaced with a modern and easy to use interface (with both a desktop version and a mobile application version) and an appropriate higher-level model should be used (i.e., ontologies) so the users will not need to quantify their preferences but present them to the system using a different approach. End-users should not be exposed to the underlying computational model. Furthermore, more advanced machine learning techniques could be applied to the new high-level model. A step towards this model has been done in the myVisitPlanner project but it can be extended further.

Concerning the system's ability to cope with a large number of users all at once, different scheduling servers could be utilized to scale the system, and the system's persistent storage should be ported from Java serializable objects to a modern SQL database.

References

[1] Achterberg, T., Berthold, T., Koch, T., and Wolter, K. Constraint integer programming: A new approach to integrate cp and mip. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Berlin, Heidelberg, 2008), CPAIOR'08, Springer-Verlag, pp. 6–20.

[2] Alexiadis, A., and Refanidis, I. Meeting the objectives of personal activity scheduling through post-optimization. First International Workshop on Search Strategies and Non-standard Objectives (SSNOWorkshop 2012). CPAIOR, Nantes, France, May 2012.

[3] Alexiadis, A., and Refanidis, I. Generating alternative plans for scheduling personal activities. In *Proceedings of ICAPS 2013 Workshop on Scheduling and Planning Applications (SPARK)* (2013), pp. 35–40.

[4] Alexiadis, A., and Refanidis, I. Post-optimizing individual activity plans through local search. In *Proceedings of ICAPS 2013 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)* (2013), pp. 7–15.

[5] Antila, V., Mäntyjärvi, J., and Könönen, V. Optimizing meeting scheduling in collaborative mobile systems through distributed voting. In *Eigth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2010, March 29 - April 2, 2010, Mannheim, Germany, Workshop Proceedings* (2010), IEEE, pp. 238–243.

[6] Apt, K. R. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[7] Apt, K. R., and Wallace, M. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007.

[8] Baker, K. R. *Elements of sequencing and scheduling*. K.R. Baker at Dartmouth College, Hanover, NH US, 2005.

[9] Bank, J., Cain, Z., Shoham, Y., Suen, C., and Ariely, D. Turning personal calendars into scheduling assistants. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts* (New York, NY, USA, 2012), CHI EA '12, ACM, pp. 2667–2672.

[10] Beck, J. C., and Fox, M. S. A generic framework for constraint-directed search and scheduling. *AI Magazine 19*, 4 (1998), 101–130.

[11] Berry, P., Conley, K., Gervasio, M., Peintner, B., Uribe, T., and Yorke-Smith, N. Deploying a personalized time management agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS06) Industrial Track* (Hakodate, Japan, May 2006), p. 1564–1571.

[12] Berry, P., Donneau-Golencer, T., Duong, K., Gervasio, M., Peintner, B., and Yorke-Smith, N. Evaluating user-adaptive systems: Lessons from experiences with a personalized meeting scheduling assistant. In *Proceedings of the Twenty-First International Conference on Innovative Applications of Artificial Intelligence (IAAI09)* (Pasadena, CA, July 2009), AAAI Press.

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

[14] Bertsimas, D., and Tsitsiklis, J. *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.

[15] Blum, A., and Furst, M. L. Fast planning through planning graph analysis. *Artificial Intelligence 90*, 1-2 (1997), 281–300.

[16] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35*, 3 (Sept. 2003), 268–308.

[17] Blythe, J. Decision-theoretic planning. *AI Magazine 20*, 2 (1999), 37–54.

[18] Boddy, M. S., Fox, M., and Thiébaux, S., Eds. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007* (2007), AAAI.

[19] Boutilier, C., Dean, T. L., and Hanks, S. Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artif. Intell. Res. (JAIR) 11* (1999), 1–94.

[20] Brachman, R. J., Levesque, H. J., and Reiter, R., Eds. *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89). Toronto, Canada, May 15-18 1989* (1989), Morgan Kaufmann.

[21] Bradley, S. P., Hax, A. C., and Magnanti, T. L. *Applied mathematical programming*. Addison-Wesley, Reading Mass., 1977.

[22] Brafman, R. I., Geffner, H., Hoffmann, J., and Kautz, H. A., Eds. *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010* (2010), AAAI.

[23] Brown, G. G., and Dell, R. F. Formulating integer linear programs: A rogues' gallery. *INFORMS Transactions on Education 7*, 2 (2007), 153–159.

[24] Burgard, W., and Roth, D., Eds. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011* (2011), AAAI Press.

[25] Card, S. K., Newell, A., and Moran, T. P. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

[26] Chandru, V., and Hooker, J. *Optimization Methods for Logical Inference*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and sons, Inc, New York, Chichester, Weinheim, 1999.

[27] Chin, J. P., Diehl, V. A., and Norman, K. L. Development of an instrument measuring user satisfaction of the human-computer interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1988), CHI '88, ACM, pp. 213–218.

[28] Chrpa, L., McCluskey, T. L., and Osborne, H. Optimizing plans through analysis of action dependencies and independencies. In McCluskey et al. [77].

[29] Coman, A., and Muñoz-Avila, H. Generating diverse plans using quantitative and qualitative plan distance metrics. In Burgard and Roth [24].

[30] Conley, K., and Carpenter, J. Towel: Towards an intelligent to-do list. In *Proceedings of AAAI Spring Symposium on Interaction Challenges for Artificial Assistants* (Stanford, CA, March 2007), AAAI.

[31] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1971), STOC '71, ACM, pp. 151–158.

[32] Crawford, J. M., and Baker, A. B. Experimental results on the application of satisfiability algorithms to scheduling problems. In Hayes-Roth and Korf [54], pp. 1092–1097.

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

[34] David, J., and Tat-Leong, C. Constraint-based applications in production planning: examples from the automative industry. In *Proceedings of the Practical Application of Constraint Technology* (Paris, 1995), pp. 35–51.

[35] Dean, T., and Kambhampati, S. Planning and scheduling. In Tucker [127], pp. 614–636.

[36] Dechter, R., Flerova, N., and Marinescu, R. Search algorithms for m best solutions for graphical models. In Hoffmann and Selman [57].

[37] Eén, N., and Sörensson, N. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation 2* (2006), 1–26.

[38] Feige, U., and Jozeph, S. Separation between estimation and approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science* (New York, NY, USA, 2015), ITCS '15, ACM, pp. 271–276.

[39] Fikes, R., and Lehnert, W. G., Eds. *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993* (1993), AAAI Press / The MIT Press.

[40] Fikes, R. E., and Nilsson, N. J. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1971), IJCAI'71, Morgan Kaufmann Publishers Inc., pp. 608–620.

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

[42] Gardi, F., Benoist, T., Darlay, J., Estellon, B., and Megel, R. *Mathematical Programming Solver Based on Local Search*, 1st ed. Wiley-IEEE Press, 2014.

[43] Garrido, L., and Sycara, K. Multi-agent meeting scheduling: Preliminary experimental results. In *2nd International Conference on Multi-agent Systems* (Kyoto, Japan, 1996), IMACS, AAAI Press, pp. 95–102.

[44] Genesereth, M. R., and Nilsson, N. J. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[45] Ginsberg, M. *Essentials of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[46] Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res. 13*, 5 (May 1986), 533–549.

[47] Glover, F., and Kochenberger, G. A., Eds. *Handbook of metaheuristics*. International series in operations research & management science. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003.

[48] Glover, F., and Laguna, M. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[49] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[50] Gomes, C. P., Kautz, H., Sabharwal, A., and Selman, B. Satisfiability solvers. In *Handbook of Knowledge Representation*, vol. 3 of *Foundations of Artificial Intelligence*. Elsevier, 2008, pp. 89–134.

[51] Greif, I. *PCAL: A Personal Calendar*. MIT/LCS/TM. M.I.T. Lab. for CompSc., 1982.

[52] Greif, I. The user interface of a personal calendar program. In *Proc. Of the NYU Symposium on User Interfaces on Human Factors and Interactive Computer Systems* (Norwood, NJ, USA, 1984), Ablex Publishing Corp., pp. 207–222.

[53] Hadjiconstantinou, E., Lucas, C., Mitra, G., and Moody, S. Tools for reformulating logical forms into zero-one mixed integer problems. *European Journal of Operational Research 72*, 2 (1994), 262–276.

[54] Hayes-Roth, B., and Korf, R. E., Eds. *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2* (1994), AAAI Press / The MIT Press.

[55] Hegery, B., Hung, C. C., and Kasprak, K. A comparative study on differential evolution and genetic algorithms for some combinatorial problems. In *Proceedings of 8th Mexican International Conference on Artificial Intelligence* (2009).

[56] Hentenryck, P. V., Saraswat, V., and Deville, Y. Design, implementation, and evaluation of the constraint language cc(fd). *Journal of Logic Programming 37* (1998), 139–164.

[57] Hoffmann, J., and Selman, B., Eds. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada* (2012), AAAI Press.

[58] Hooker, J. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and sons, Inc, New York, Chichester, Weinheim, 2000.

[59] Hooker, J. Operations research methods in constraint programming. In *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, ch. 15.

[60] Hooker, J. N. Good and bad futures for constraint programming (and operations research). *Constraint Programming Letters 1* (2007), 21–32.

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

[62] Jennings, N. R., and Jackson, A. J. Agent based meeting scheduling: A design and implementation. *IEE Electronic Letters 18*, 5 (1995), 350–352.

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

[64] Kambhampati, S. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models, 2007.

[65] Kernighan, B. W., and Ritchie, D. M. *The C Programming Language*, 2nd ed. Prentice Hall Professional Technical Reference, 1988.

[66] Kieras, D. Towards a practical goms model methodology for user interface design. *Handbook of Human-Computer Interaction* (1988), 135–137.

[67] Kimbal Marriott, P. S. *Programming with Constraints: An Introduction*. MIT, 1998.

[68] Kincaid, C. M., Dupont, P. B., and Kaye, A. R. Electronic calendars in the office: An assessment of user needs and current technology. *ACM Trans. Inf. Syst. 3*, 1 (Jan. 1985), 89–102.

[69] Kirakowski, J., and Corbett, M. Sumi: the software usability measurement inventory. *British Journal of Educational Technology 24*, 3 (1993), 210–212.

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

[71] Klee, V., and Minty, G. J. How good is the simplex algorithm? In *Inequalities III*, O. Shisha, Ed. Academic Press Inc., New York, 1972, pp. 159–175.

[72] Kumar, V. Algorithms for constraint satisfaction problems: A survey. *AI Magazine 13*, 1 (1992), 32–44.

[73] Land, A. H., and Doig, A. G. An automatic method of solving discrete programming problems. *Econometrica 28*, 3 (1960), 497–520.

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

[75] Lewis, H. R., and Papadimitriou, C. H. *Elements of the Theory of Computation*, 2nd ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[76] Likas, A., Blekas, K., and Kalles, D., Eds. *Artificial Intelligence: Methods and Applications - 8th Hellenic Conference on AI, SETN 2014, Ioannina, Greece, May 15-17, 2014. Proceedings* (2014), vol. 8445 of *Lecture Notes in Computer Science*, Springer.

[77] McCluskey, L., Williams, B., Silva, J. R., and Bonet, B., Eds. *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012* (2012), AAAI.

[78] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. PDDL—The Planning Domain Definition Language—Version 1.2. Tech. rep., Yale Center for Computational Vision and Control, 1998.

[79] Milano, M., Ed. *Constraint and integer programming : toward a unified methodology*. Operations research/computer science interfaces series. Kluwer Academic Publishers, Boston, London, 2004.

[80] Milano, M., and Hentenryck, P. V. *Hybrid Optimization - The Ten Years of CPAIOR*, vol. 45 of *Springer Optimization and its Applications*. Springer, 2011.

[81] Minton, S., Philips, A., Johnston, M. D., and Laird, P. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *Journal of Artificial Intelligence Research 58* (1993), 161–205.

[82] Modi, P. J., Veloso, M., Smith, S., and Oh, J. CMRadar: A Personal Assistant Agent for Calendar Management. In *Agent-Oriented Information Systems II, Lecture Notes in Computer Science*, vol. 3508/2005. Springer-Verlag, 2005, pp. 169–181.

[83] Montanari, U. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences 7* (Jan. 1974), 95–132.

[84] Myers, K. Building an intelligent personal assistant. AAAI Invited Talk, 2006.

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

[86] Myers, K. L., and Lee, T. J. Generating qualitatively different plans through metatheoretic biases. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence* (Menlo Park, CA, USA, 1999), AAAI '99/IAAI '99, American Association for Artificial Intelligence, pp. 570–576.

[87] Nakhost, H., and Müller, M. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In Brafman et al. [22], pp. 121–128.

[88] Nau, D. S., Gupta, S. K., and Regli, W. C. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes* (1995), pp. 1670–1676.

[89] Nelder, J., and Mead, R. A simplex method for function minimization. *The Computer Journal 7*, 4 (1965), 308.

[90] Nemhauser, G. L., and Wolsey, L. A. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[91] Newell, A., Shaw, J. C., and Simon, H. A. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing* (1959), pp. 256–264.

[92] Nguyen, T. A., Do, M. B., Gerevini, A., Serina, I., Srivastava, B., and Kambhampati, S. Generating diverse plans to handle unknown and partially known user preferences. *Artif. Intell. 190* (2012), 1–31.

[93] Niederlinski, A. *A Gentle Guide to Constraint Logic Programming via ECLiPSe*, 3rd ed. Jacek Skalmierski Computer Studio, PL-44-100 Gliwice, 2014.

[94] Nielsen, J., and L. Mack, R. *Usability Inspection Methods*. John Wiley Sons, 1994.

[95] Norvig, P. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[96] Palen, L. 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* (New York, NY, USA, 1999), CHI '99, ACM, pp. 17–24.

[97] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[98] Payne, T. R., Singh, R., and Sycara, K. P. Calendar agents on the semantic web. *IEEE Intelligent Systems 17*, 3 (2002), 84–86.

[99] Pednault, E. P. D. Adl: Exploring the middle ground between strips and the situation calculus. In Brachman et al. [20], pp. 324–332.

[100] Pinedo, M. *Scheduling : theory, algorithms, and systems*. Upper Saddle, N.J. Prentice Hall, 2002.

[101] Refanidis, I. Managing personal tasks with time constraints and preferences. In Boddy et al. [18], pp. 272–279.

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

[103] Refanidis, I., Emmanouilidis, C., Sakellariou, I., Alexiadis, A., Koutsiamanis, R.-A., Agnantis, K., Tasidou, A., Kokkoras, F., and Efraimidis, P. S. myvisitplanner^{gr}: Personalized itinerary planning system for tourism. In Likas et al. [76], pp. 615–629.

[104] Refanidis, I., McCluskey, T. L., and Dimopoulos, Y. Planning services for individuals: A new challenge for the planning community. In *In Workshop on Connecting Planning Theory with Practice* (2004), pp. 56–61.

[105] Refanidis, I., and Yorke-Smith, N. A constraint-based approach to scheduling an individual's activities. *ACM Trans. Intell. Syst. Technol. 1*, 2 (Dec. 2010), 12:1–12:32.

[106] Rich, E., and Knight, K. *Artificial Intelligence*, 2nd ed. McGraw-Hill Higher Education, 1990.

[107] Roberts, M., Howe, A., Ray, I., and Urbanska, M. Using planning for a personalized security agent. AAAI Workshops, 2012.

[108] Rodosek, R., Wallace, M., and Hajian, M. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research 86*, 0 (1999), 63–87.

[109] Rossi, F., Petrie, C., and Dhar, V. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence* (1990), pp. 550–556.

[110] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[111] Schimpf, J., and Shen, K. Eclipse — from lp to clp. *Theory Pract. Log. Program. 12*, 1–2 (Jan. 2012), 127–156.

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

[113] Schrijver, A. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[114] Schrijver, A. *Combinatorial Optimization: Polyhedra and Efficiency (3 volume, A,B, C)*, 1 ed. Algorithms and Combinatorics 24. Springer, 2003.

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

[116] Sen, S., and Durfee, E. H. On the design of an adaptive meeting scheduler. In *Proc. of the Tenth IEEE Conference on AI Applications* (1994), pp. 40–46.

[117] Sen, S., and Durfee, E. H. A formal study of distributed meeting scheduling. *Group Decision and Negotiation 7* (1998), 265–289.

[118] Sen, S., Haynes, T., and Arora, N. Satisfying user preferences while negotiating meetings. *Intl. Journal on Human-Computer Studies 47*, 3 (1997), 407–427.

[119] Shen, K., and Schimpf, J. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In van Beek [128], pp. 622–636.

[120] Simon, H. A., and Newell, A. Human problem solving: The state of the theory in 1970. *American Psychologist 26*, 2 (Feb 1971), 145–159.

[121] Singh, R., and Sycara, K. Rcal: An autonomous agent for intelligent distributed meeting scheduling. In *Technical Report CMU-RI-TR-03-46* (2003).

[122] Smith, D. E., Frank, J., and Jónsson, A. K. Bridging the gap between planning and scheduling. *Knowledge Engineering Review 15* (2000), 2000.

[123] Smith, S. F., and Cheng, C.-C. Slack-based heuristics for constraint satisfaction scheduling. In Fikes and Lehnert [39], pp. 139–144.

[124] Sterling, L., and Shapiro, E. *The Art of Prolog: Advanced Programming Techniques*, 2 ed. MIT Press, Cambridge, MA, USA, 1994.

[125] Storn, R., and Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization 11*, 4 (Dec. 1997), 341–359.

[126] Tate, A., Drabble, B., and Kirby, R. O-plan2: an open architecture for command, planning and control. In *Intelligent Scheduling* (1994), Morgan Kaufmann, pp. 213–239.

[127] Tucker, A. B., Ed. *The Computer Science and Engineering Handbook*. CRC Press, 1997.

[128] van Beek, P., Ed. *Principles and Practice of Constraint Programming — CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1–5, 2005, Proceedings* (2005), vol. 3709 of *Lecture Notes in Computer Science*, Springer.

[129] Van Hentenryck, P. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, USA, 1989.

[130] Weld, D. S. An introduction to least commitment planning. *AI Magazine 15*, 4 (1994), 27–61.

[131] Weld, D. S. Recent advances in ai planning. *AI Magazine 20*, 2 (1999), 93–123.

[132] Wilkins, D. E. Can ai planners solve practical problems? *Computational Intelligence 6*, 4 (1990), 232–246.

[133] Williams, H. P. *Model Solving in Mathematical Programming*. Wiley and Sons, 1993.

[134] Williams, H. P. *Model Building in Mathematical Programming*, 5 ed. Wiley, 2013.

[135] Winston, P. H., and Horn, B. K. P. *Lisp*. Reading, Mass. Addison-Wesley, 1989.

[136] Wolfram, S. *The Mathematica Book*, 5 ed. Wolfram Media, Incorporated, 2003.

APPENDIX A

Code & Setup Instructions

All the source code written for this thesis is available online.[1] For setting up SelfPlanner you will need a UNIX/Linux system with Oracle Java Enterprise 7+, Apache and PHP 5.3.[2] Create an account for SelfPlanner and follow the instructions below.

A.1 Contents of *thesis-code.tar.gz*

- Branch and Bound solver in ECL^{i}PS^e Prolog : `scheduler/clp-bb/default-heuristic`
- Modified Brand and Bound solver in ECL^{i}PS^e Prolog with custom heuristic : `scheduler/clp-bb/custom-heuristic`
- Local search solver in ECL^{i}PS^e Prolog : `scheduler/clp-custom`
- Genetic Algorithm solver in C : `scheduler/genalg`
- Proposed scheduler based on SWO and Simulated Annealing with Tabu Lists and backtracking in C++ : `scheduler/ final`
- Part of the SelfPlanner client in HTML/Javascript/PHP : `selfplanner/web`
- SelfPlanner client in Java : `selfplanner/src/myplanner-app`
- SelfPlanner server in Java : `selfplanner/src/myplanner`
- SelfPlanner API library in Java : `selfplanner/src/myplanner-api`
- SelfPlanner Model UML diagrams : `selfplanner/uml`
- SelfPlanner binaries : `selfplanner/bin`

[1] `http://java.uom.gr/~talex/thesis-code.tar.gz`

[2] The SelfPlanner PHP code is incompatible with the newer releases of PHP. If you want to use a newer PHP version you will have to port the PHP code.

- Reference *Combined Domain* implementation in Java : `selfplanner/joint-activity-domains`

A.2 Setting up the Client

1. Create the folders `~/maps` and `~/public_html` and `~/comments`.
2. Give owner permissions to the PHP account for the folders maps and comments, e.g., `chown php maps, chmod 775 maps*`
3. Copy the contents of `thesis-code/selfplanner/web` to your `~/public_html` folder.
4. Edit `checklogin.php/checklogin2.php/checklogin3.php` and `auth.php` and change `$file = "/home/selfplanner/…` to your home folder.
5. Edit `sp.html` and change `<param name=plannerAddr VALUE = "195.251.210.248">` to your server's IP address.
6. Connect to Google Developer Console and generate a key for the Google MAPS API. Edit `maps.php` and change the line `<script src="http://maps.google.com/maps?file=api&v=2&sensor=false&key=AIzaSyAs9Tf9Zl3xYDdjsXD6GpSYpIJBZ58UTdg" type="text/javascript"></script>` so that it uses your new key.
7. Copy `thesis-code/selfplanner/bin/myplanner-app.jar` to `~/public_html` and sign it with jarsigner and a certificate.

A.3 Setting up the Server

1. Create the folder `~/google`.
2. Download the Google Calendar API for Java library and set it in your classpath.
3. Copy `thesis-code/selfplanner/bin/myplanner.jar,sp,spasswd` to your home folder.
4. Generate `sp-host`, in your home folder, with your system's hostname.
5. Register your SelfPlanner installation with Google Developer console and generate a json file for the OAuth API for callback calls by Google. First go to APIs and enable the Google Calendar API. Then you will have to go to Credentials ⇒ OAuth ⇒ Create new Client ID. Specify `http://<your-hostname>:8081` for `Javascript Origins` and `http://<your-hostname>:8081/Callback` for `Redirect URIs`. Download the json file and rename it to `google.json`. You will have to copy this file to the installation's home folder.
6. Copy `thesis-code/scheduler/final/*` to the installation's home folder. Compile the scheduler with `g++ -O3 -mtune=native -o swo swo.cpp swo.h hillc.cpp hillc.h`.
7. You can start SelfPlanner with `nohup ./sp &`. The log-file is `sp-log`.

A.4 Setting up the API

1. Compile `thesis-code/selfplanner/source/myplanner-api` to generate `myplanner-api.jar`.

2. Run `java -cp myplanner-api.jar gr.uom.csse.ai.myplanner.CryptoManager`. This will generate a valid keypair, copy-paste the private key to a safe place—you will use it to connect to SelfPlanner from your application. Copy the generated `mvp-key.sp` file to SelfPlanner server's home folder and restart the SelfPlanner server.

3. You can now call SelfPlanner from your application using the `myplanner-api.jar` Java library and the private key.

APPENDIX B

Using SelfPlanner through its API—An Example

Let us define three activities, two located at the university, and one at the town center. One will be interruptible, one non-interruptible and the final one will non-interruptible periodic. Completing the non-interruptible one before the interruptible own is a strong preference, but not a necessity.

Listing B.1: Using The SelfPlanner API Example in Java

```
// Initialize the data structures
Console console = new Console() {
    @Override
    public void write(String s) {
        System.out.println("SelfPlanner: " + s);
    }
};
int id = 100;
Date now = new Date();
long ts = now.getTime();
TaskManager tmgr = new TaskManager("" + id + "_" + ts);
LocationManager lmgr = new LocationManager("" + id + "_" + ts);
BinaryManager ordconstraints = new BinaryManager("" + id + "_" + ts, tmgr);
TemplateManager templates = new TemplateManager("" + id + "_" + ts);
DomainManager dmg = new DomainManager("" + id + "_" + ts);
LocationClassManager lcmgr = new LocationClassManager("" + id + "_" + ts);
BinaryManager dminconstraints = new BinaryManager("" + id + "_" + ts, tmgr);
BinaryManager dmaxconstraints = new BinaryManager("" + id + "_" + ts, tmgr);
BinaryManager iconstraints = new BinaryManager("" + id + "_" + ts, tmgr);
MyPlannerData data = new MyPlannerData("" + id + "_" + ts, tmgr, lmgr, ordconstraints,
    templates, dmg, lcmgr, dminconstraints, dmaxconstraints, iconstraints);
// Create the clear domain template
Template clearDomain = new Template(DomainAction.ActionAffects.DAY, "Clear Domain", -1);
clearDomain.setMultipleActions(0, DomainAction.Action.NOT_INC, 48);
// Define the problem instance
// Locations
```

```
26 Location univ = new Location(0, "University");
27 univ.setLatLng("40.62554905578553␣22.960052490234375");
28 lmgr.addLocation(univ);
29 Location center = new Location(1,"Town␣Center");
30 center.setLatLng("40.626330777594895␣22.9489803314209");
31 lmgr.addLocation(center);
32 // Get the real distance using the Google Distance Matrix API
33 LocationUtils.createLocationPairUsingDistanceMatrix(univ, center, lmgr);
34 try {
35     Thread.sleep(30);
36 } catch (InterruptedException e) {}
37 LocationUtils.createLocationPairUsingDistanceMatrix(center, univ, lmgr);
38 // Enable Traveling Time Minimization
39 data.setLocationDistanceUtility(LOCATION_DISTANCE_UTILITY);
40 // Interruptible Activity
41 Domain d1 = new Domain(startDate, endDate);
42 d1.addTemplate(clearDomain, null, null, DomainAction.Action.NOT_INC);
43 for (Date[] interval : intervalsOfActivity1) {
44     d1.addManualAction(new ManualAction(interval[0],
45         interval[1], DomainAction.Action.INC));
46 }
47 DomainPrefs prefs1 = new NewDomainPrefs(NewDomainPrefs.NLPref.MORNING, TEMPORAL_UNARY_UTIL);
48 Task t1 = new Task(0, "Work␣on␣project", 150, 300, extraDurationUtil1, "University", d1,
       prefs1, null, activityUtility1, 1.0, 30, 90, 180, 1440);
49 tmgr.addTask(t1);
50 // Non-Interruptible Activity
51 Domain d2 = new Domain(startDate, endDate);
52 d2.addTemplate(clearDomain, null, null, DomainAction.Action.NOT_INC);
53 for (Date[] interval : intervalsOfActivity2) {
54     d2.addManualAction(new ManualAction(interval[0],
55         interval[1], DomainAction.Action.INC));
56 }
57 DomainPrefs prefs2 = new NewDomainPrefs(NewDomainPrefs.NLPref.AFTERNOON, TEMPORAL_UNARY_UTIL
       );
58 Task t2 = new Task(1, "Visit␣library", 120, 180, extraDurationUtil2, "Town␣Center", d1,
       prefs2, null, activityUtility2, 1.0);
59 tmgr.addTask(t2);
60 // Periodic Activity
61 Domain d3 = new Domain(startDate, endDate);
62 d3.addTemplate(clearDomain, null, null, DomainAction.Action.NOT_INC);
63 for (Date[] interval : intervalsOfActivity3) {
64     d3.addManualAction(new ManualAction(interval[0],
65         interval[1], DomainAction.Action.INC));
66 }
67 DomainPrefs prefs3 = new NewDomainPrefs(NewDomainPrefs.NLPref.AFTERNOON, TEMPORAL_UNARY_UTIL
       );
68 PeriodicPrefs.Period period = PeriodicPrefs.Period.DAILY;
69 PeriodicPrefs pprefs = new PeriodicPrefs(period, PeriodicPrefs.Range.END_BY_DEADLINE, true,
       true);
70 Task t3 = new Task(2, "Have␣lunch", 30, 60, extraDurationUtil1, "University", d1, prefs3,
       pprefs, activityUtility3, 1.0);
71 tmgr.addTask(t3);
72 BinaryPreference p1 = new BinaryPreference(1, 0, binaryUtil, BinaryPreference.ORD);
73 ordconstraints.addPreference(p1);
```

```
// Call SelfPlanner to solve the problem instance
SelfPlannerClient client =  new SelfPlannerClient("oaristotelis.uom.gr", console, PRIV_KEY,
    data, "-k 7000");
Solution[] solutions = null;
TaskSolved[] solved;
try {
    solutions = client.simpleConnectAndSolve(NUM_OF_PLANS, univ); // The current location is
         at the university
} catch (TooManyClientsException ex) {
    System.out.println("SelfPlanner overload. Exiting.");
    System.exit(1);
}
if (solutions == null || solutions.length == 0)
    System.out.println("No solution was returned");
else { // Print Plans
    int pn,pn2;
    for (int i=0;i<NUM_OF_PLANS;i++) {
        solved = solutions[i].tasks();
        pn = i + 1;
        System.out.println("Printing plan " + pn);
        for (TaskSolved element : solved) {
            for (int j=0;j<element.solution().length;j++) {
                for (int k=0;k<element.solution()[j].length;k++) {
                    pn = j + 1;
                    pn2 = k + 1;
                    System.out.println(element.task().name() + ":Period " + pn + "," + ", 
                        part " + pn2);
                    System.out.println(element.solution()[j][k].getStartTime() + " : " +
                        element.solution()[j][k].getEndTime());
                }
            }
        }
    }
}
```

APPENDIX C

The Complete Solver in ECL^{i}PS^{e} Prolog

Listing C.1: complete-solver.ecl

```
:-lib(ic).
:-lib(ic_global).
:-lib(ic_kernel).
:-lib(branch_and_bound).
:-lib(propia).

:-dynamic before/2.
:-dynamic min_dist/3.
:-dynamic max_dist/3.
:-dynamic implies/2.
:-dynamic before/3.
:-dynamic min_dist/4.
:-dynamic max_dist/4.
:-dynamic implies/3.
:-dynamic loc_dist/3.

task(id(0), task_locs(1), loc([1]), dur(9,9,0.000000,1), smin(2), smax(6), prox_cons
    (5,2147483647), prox_prefs(0,0.000000,2147483647,0.000000), temporal_pref(-1,0,5.000000)
    , utility(9.000000), utilization(1.000000), domain([9..14, 29..59, 79..117, 120..156,
    162..197, 204..207, 213..227, 237..259, 266..289, 300..316, 334..341, 349..381, 392..
    426, 430..436, 454..457, 459..490])).
task(id(1), task_locs(1), loc([1]), dur(5,5,0.000000,0), smin(5), smax(5), prox_cons
    (0,2147483647), prox_prefs(0,0.000000,2147483647,0.000000), temporal_pref(-1,0,9.000000)
    , utility(7.000000), utilization(1.000000), domain([40..77, 96..109, 121..156, 159..173,
     184..195, 208..221, 228..264, 272..303, 314..347, 350..385, 397..421, 430..454, 461..
    469, 472..477, 482..494])).
task(id(2), task_locs(3), loc([1]), dur(9,12,4.000000,1), smin(2), smax(8), prox_cons
    (7,2147483647), prox_prefs(0,0.000000,2147483647,0.000000), temporal_pref(0,0,0.000000),
     utility(5.000000), utilization(1.000000), domain([10..38, 39..57, 59..79, 91..93, 98..
    109, 116..119, 120..130, 133..168, 174..194, 211..247, 251..271, 278..293, 307..311, 315
    ..323, 335..354, 366..377, 381..419, 427..438, 445..460, 468..486])).
task(id(3), task_locs(4), loc([1]), dur(6,6,0.000000,1), smin(2), smax(3), prox_cons(8,96),
    prox_prefs(0,0.000000,2147483647,0.000000), temporal_pref(1,0,6.000000), utility
    (11.000000), utilization(1.000000), domain([4..18, 31..60, 76..113, 123..151, 165..179,
```

```
    191..213, 223..229, 245..253, 257..288, 296..304, 314..341, 357..394, 397..417, 427..
    435, 455..464, 475..486])).
task(id(4), task_locs(1), loc([1]), dur(8,8,0.000000,1), smin(2), smax(6), prox_cons
    (5,2147483647), prox_prefs(0,0.000000,70,5.000000), temporal_pref(1,0,5.000000), utility
    (8.000000), utilization(1.000000), domain([18..37, 51..88, 94..120, 121..138, 144..147,
    149..170, 185..213, 225..250, 267..293, 298..329, 345..381, 382..412, 417..427, 441..
    450, 460..495])).

nof_locations(4).
loc_dist(0,0,0).
loc_dist(0,1,0).
loc_dist(0,2,0).
loc_dist(0,3,0).
loc_dist(1,0,0).
loc_dist(1,1,0).
loc_dist(1,2,4).
loc_dist(1,3,3).
loc_dist(2,0,0).
loc_dist(2,1,4).
loc_dist(2,2,0).
loc_dist(2,3,5).
loc_dist(3,0,0).
loc_dist(3,1,3).
loc_dist(3,2,5).
loc_dist(3,3,0).
ordering_constraints(2).
before(1,4).
before(3,4).
min_dist_constraints(3).
min_dist(1,4,5).
min_dist(2,3,10).
min_dist(2,4,8).
max_dist_constraints(0).
implication_constraints(2).
implies(2,3).
implies(4,3).
ordering_preferences(2).
%before(0,4,1.000000).
%before(1,2,3.000000).
%min_dist_preferences(2).
%min_dist(1,4,10,1.000000).
%min_dist(3,4,9,1.000000).
%max_dist_preferences(0).
%implication_preferences(2).
%implies(0,1,2.000000).
%implies(1,4,2.000000).

post_utilization_constraints(Tasks,MinT,MaxT):-
    Length is MaxT-MinT,
    length(TimePoints,Length),
    TimePoints#::0..1000,
    post_utilization_constraints1(MinT,MaxT,TimePoints,Tasks).

post_utilization_constraints1(MaxT,MaxT,[],_Tasks):-!.
```

```
post_utilization_constraints1(T,MaxT,[Point|TimePoints],Tasks):-
    post_utilization_constraints2(T,Tasks,Point),
    T1 is T+1,
    post_utilization_constraints1(T1,MaxT,TimePoints,Tasks).

post_utilization_constraints2(_T,[],0).
post_utilization_constraints2(T,[task(ID,Subtasks,Flag,_Duration)|Tasks],Point):-
    post_utilization_constraints3(T,ID,Subtasks,PointCurrent,Flag),
    Flag#=1 => Point#=PointCurrent+PointNext,
    Flag#=0 => Point#=PointNext,
    post_utilization_constraints2(T,Tasks,PointNext).

post_utilization_constraints3(_T,_,[],0, _Flag).
post_utilization_constraints3(T,ID,[subtask(TS,DS,_L)|Subtasks],
            PointCurrent,Flag):-
    task(id(ID),_,_,_,_,_,_,_,_,_,utilization(Utilization),_),
    Uz is integer(Utilization*1000),
    (Flag#=1 and TS#=<T and (TS+DS)#>T) => PointCurrent#=Uz+PointNext,
    (Flag#=0 or TS#>T or (TS+DS)#=<T) => PointCurrent#=PointNext,
    post_utilization_constraints3(T,ID,Subtasks,PointNext,Flag).

createlocsm(Locs,Locs).
createlocsm(Locs,Cnt):-
    (not loc_dist(Cnt,-1,-1), assert(loc_dist(Cnt,-1,-1)); writeln('assertion␣exists')),
    (not loc_dist(-1,Cnt,-1), assert(loc_dist(-1,Cnt,-1)); writeln('assertion␣exists')),
    Cnt1 is Cnt+1,
    createlocsm(Locs,Cnt1).

go(Tasks):-
    (not loc_dist(-1,-1,-1), assert(loc_dist(-1,-1,-1)); writeln('assertion␣exists')),
    nof_locations(Locations_size),
    createlocsm(Locations_size,0),
    findall(task(ID,_Subtasks,_Flag,_Duration),
    task(id(ID),_,_,_,_,_,_,_,_,_,_),Tasks),
    time_points(Tasks,MinT,MaxT),
    post_domains(Tasks),
    findall(loc_dist(Loc1,Loc2,LocDist),loc_dist(Loc1,Loc2,LocDist),LocDistances),
    post_distances(Tasks,LocDistances),
    post_utilization_constraints(Tasks,MinT,MaxT),
    post_unary_proximity_constraints(Tasks),
    findall(before(ID1,ID2),before(ID1,ID2),OrderingConstraints),
    post_ordering_constraints(OrderingConstraints,Tasks),
    findall(min_dist(ID1,ID2,Dist),min_dist(ID1,ID2,Dist),MinDistConstraints),
    post_mindist_constraints(MinDistConstraints,Tasks),
    findall(max_dist(ID1,ID2,Dist),max_dist(ID1,ID2,Dist),MaxDistConstraints),
    post_maxdist_constraints(MaxDistConstraints,Tasks),
    findall(implies(ID1,ID2),implies(ID,ID2),ImplicationConstraints),
    post_implication_constraints(ImplicationConstraints,Tasks),
    findall(before(ID1,ID2,PrefUtil),before(ID1,ID2,PrefUtil),OrderingPreferences),
    compute_ordering_preferences(OrderingPreferences,Tasks,OrderingUtil),
    findall(min_dist(ID1,ID2,Dist,PrefUtil),min_dist(ID1,ID2,Dist,PrefUtil),
        MinDistPreferences),
    compute_mindist_preferences(MinDistPreferences,Tasks,MinDistUtil),
```

```
    findall(max_dist(ID1,ID2,Dist,PrefUtil),max_dist(ID1,ID2,Dist,PrefUtil),
        MaxDistPreferences),
    compute_maxdist_preferences(MaxDistPreferences,Tasks,MaxDistUtil),
    findall(implies(ID1,ID2,PrefUtil),implies(ID,ID2,PrefUtil),ImplicationPreferences),
    compute_implication_preferences(ImplicationPreferences,Tasks,ImplicationUtil),
    get_all_variables(Tasks,Variables),
    compute_objective_variable(Tasks,MinT,MaxT,Obj),
    Obj1$=0-(Obj+OrderingUtil+MinDistUtil+MaxDistUtil+ImplicationUtil),
    write(Obj1),
    !,
    %bb_min(search(Variables, 0, occurrence, indomain_random, complete, []),Obj1,
    %    bb_options(continue, -1.0Inf, 1.0Inf, 1, 1, 0, 1200, _78313, _78314)).
    bb_min(search(Variables, 0, most_constrained, indomain_split, complete, []),Obj1,
                                bb_options(continue, -1.0Inf, 1.0Inf, 1, 1, 0, 1200, _78313,
                                     _78314)).

time_points([],1000000,0).
time_points([task(ID,_Subtasks,_Flag,_Duration)|Tasks],MinT,MaxT):-
    task(id(ID),_,_,_,_,_,_,_,_,_,domain(Domain)),
    time_points2(Domain, MinT1, MaxT1),
    MinT#=min(MinT0, MinT1),
    MaxT#=max(MaxT0, MaxT1),
    time_points(Tasks, MinT0, MaxT0).

time_points2([],1000000,0).
time_points2([A..B|Domain],MinT,MaxT):-
    MinT#=min(A, MinT0),
    MaxT#=max(B, MaxT0),
    time_points2(Domain, MinT0, MaxT0).

post_unary_proximity_constraints([]):-!.
post_unary_proximity_constraints([task(ID,_Subtasks,_Flag,_Duration)|Tasks]):-
    task(id(ID),_,_,dur(_, _, _, 0), _, _, _, _, _, _, _, _),
    !,
    post_unary_proximity_constraints(Tasks).
post_unary_proximity_constraints([task(ID,Subtasks,_Flag,_Duration)|Tasks]):-
    task(id(ID),_,_,dur(_, _, _, 1), _, _, prox_cons(MinDist, MaxDist), _, _, _, _, _),
    post_unary_proximity_constraints2(Subtasks,MinDist,MaxDist),
    post_unary_proximity_constraints(Tasks).

post_unary_proximity_constraints2([],_MinDist,_MaxDist):-!.
post_unary_proximity_constraints2([_],_MinDist,_MaxDist):-!.
post_unary_proximity_constraints2([subtask(T,D,L)|Subtasks],MinDist,MaxDist):-
    post_unary_proximity_constraints3(subtask(T,D,L), Subtasks,MinDist,MaxDist),
    post_unary_proximity_constraints2(Subtasks,MinDist,MaxDist).

post_unary_proximity_constraints3(_, [],_MinDist,_MaxDist):-!.
post_unary_proximity_constraints3(subtask(T1,D1,L1),[subtask(T2,D2,_L2)|Subtasks],MinDist,
    MaxDist):-
    (D1#>0 and D2#>0) => T2#>=T1+D1+MinDist or T1#>T2+D2+MinDist,
    (D1#>0 and D2#>0) => T1+MaxDist#>=T2+D2,
    (D1#>0 and D2#>0) => T2+MaxDist#>=T1+D1,
    post_unary_proximity_constraints3(subtask(T1,D1,L1),Subtasks,MinDist,MaxDist).
```

```
compute_objective_variable([],_MinT,_MaxT,0):-!.
compute_objective_variable([task(ID,Subtasks,Flag,Duration)|Tasks],MinT,MaxT,Obj1):-
    task(id(ID),_,_,dur(DurMin,DurMax,Durutil,_),_,_,_,prox_prefs(MinDistPref,
        MinProxPrefUtil,MaxDistPref,MaxProxPrefUtil),temporal_pref(Pref,Point,TUtil),utility
        (U),_,_),
    compute_durutil(DurMin,DurMax,Durutil,Duration,UtilDur),
    compute_temporal_pref1(Subtasks,Flag,Duration,Pref,Point,TUtil,MinT,MaxT,TempUtil),
    compute_unary_proximity_preferences2(Subtasks,MinDistPref,MaxDistPref,ProxMinUtil,
        ProxMaxUtil,Tot),
    (Duration#=0 or Tot#=0) => ProxMinU$=0,
    (Duration#=0 or Tot#=0) => ProxMaxU$=0,
    (Duration#>0 and Tot#>0) => ProxMinU$=MinProxPrefUtil*(ProxMinUtil/Tot),
    (Duration#>0 and Tot#>0) => ProxMaxU$=MaxProxPrefUtil*(ProxMaxUtil/Tot),
    Obj1$=Obj+Flag*(U+UtilDur+TempUtil+ProxMinU+ProxMaxU),
    compute_objective_variable(Tasks,MinT,MaxT,Obj).

compute_unary_proximity_preferences2([],_MinDist,_MaxDist,0,0,0):-!.
compute_unary_proximity_preferences2([_],_MinDist,_MaxDist,0,0,0):-!.
compute_unary_proximity_preferences2([subtask(T,D,L)|Subtasks],MinDist,MaxDist,MinUtil,
    MaxUtil,Tot):-
    compute_unary_proximity_preferences3(subtask(T,D,L), Subtasks,MinDist,MaxDist,MinUtil1,
        MaxUtil1,Tot2),
    MinUtil$=MinUtil1+MinUtil2,
    MaxUtil$=MaxUtil1+MaxUtil2,
    Tot#=Tot1+Tot2,
    compute_unary_proximity_preferences2(Subtasks,MinDist,MaxDist,MinUtil2,MaxUtil2,Tot1).

compute_unary_proximity_preferences3(_, [],_MinDist,_MaxDist,0,0,0):-!.
compute_unary_proximity_preferences3(subtask(T1,D1,L1),[subtask(T2,D2,_L2)|Subtasks],MinDist
    ,MaxDist,MinUtil,MaxUtil,Tot):-
    (D1#>0 and D2#>0) => D#=MinDist-(T2-(T1+D1)),
    (D1#>0 and D2#>0) => DT#=T2+D2-T1-MaxDist,
    (D1#>0 and D2#>0 and MinDist#=<(T2-(T1+D1))) => MinUtil$=MinUtil1+D1*D2,
    (D1#=0 or D2#=0 or MinDist#>=(T2+D2-T1)) => MinUtil$=MinUtil1,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and D2#=<(D2-(D-D1)))
         => D2S$=D2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and D2#>(D2-(D-D1)))
        => D2S$=D2-(D-D1),
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and ((D1-D)*D2)#>0)
        => TMIN$=(D1-D)*D2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and ((D1-D)*D2)#=<0)
        => TMIN$=0,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and (D1-D)#>0 and (D2
        -D)#>0) => MinUtil$=MinUtil1+TMIN+D2*(D2-1)/2-(D2-D)*(D2-D-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and (D1-D)#>0 and (D2
        -D)#=<0) => MinUtil$=MinUtil1+TMIN+D2*(D2-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and (D1-D)#=<0 and
        D2S#>D1) => MinUtil$=MinUtil1+TMIN+D2S*(D2S-1)/2-(D2S-D1)*(D2S-D1-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and (D1-D)#=<0 and
        D2S#=<D1) => MinUtil$=MinUtil1+TMIN+D2S*(D2S-1)/2,
    (D1#>0 and D2#>0 and MaxDist#>=(T2+D2-T1)) => MaxUtil$=MaxUtil1+D1*D2,
    (D1#=0 or D2#=0 or MaxDist#=<(T2-(T1+D1))) => MaxUtil$=MaxUtil1,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and D2#=<(D2-(DT-D1))
        ) => D2ST$=D2,
```

```
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and D2#>(D2-(DT-D1)))
         => D2ST$=D2-(DT-D1),
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and ((D1-DT)*D2)#>0)
        => TMAX$=(D1-DT)*D2,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and ((D1-DT)*D2)#=<0)
         => TMAX$=0,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and (D1-DT)#>0 and (
        D2-DT)#>0) => MaxUtil$=MaxUtil1+TMAX+D2*(D2-1)/2-(D2-DT)*(D2-DT-1)/2,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and (D1-DT)#>0 and (
        D2-DT)#=<0) => MaxUtil$=MaxUtil1+TMAX+D2*(D2-1)/2,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and (D1-DT)#=<0 and
        D2ST#>D1) => MaxUtil$=MaxUtil1+TMAX+D2ST*(D2ST-1)/2-(D2ST-D1)*(D2ST-D1-1)/2,
    (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and (D1-DT)#=<0 and
        D2ST#=<D1) => MaxUtil$=MaxUtil1+TMAX+D2ST*(D2ST-1)/2,
    Tot#=Tot1+D1*D2,
    compute_unary_proximity_preferences3(subtask(T1,D1,L1),Subtasks,MinDist,MaxDist,MinUtil1
        ,MaxUtil1,Tot1).

compute_temporal_pref1([],_Flag,_Duration,_Pref,_Point,_TUtil,_MinT,_Maxt,0):-!.
compute_temporal_pref1([subtask(T,D,_L)|Subtasks],Flag,Duration,Pref,Point,TUtil,MinT,MaxT,
    Util):-
    compute_temporal_pref2(T,D,MinT,MaxT,Pref,Point,Util1),
    T#>=0 => Util$=((Util1*TUtil)/Duration)+Util2,
    T#= -1 => Util$=Util2,
    compute_temporal_pref1(Subtasks,Flag,Duration,Pref,Point,TUtil,MinT,MaxT,Util2).

compute_temporal_pref2(_T,_D,_MinT,_MaXT,0,_Point,0):-!.
compute_temporal_pref2(T,D,MinT,MaxT,1, _Point,Util):-
    Util$=1.01*D*(((T+D)-MinT)/(MaxT-MinT)+(T-MinT)/(MaxT-MinT))/2.
compute_temporal_pref2(T,D,MinT,MaxT,-1,_Point,Util):-
    Util$=1.01*D*((MaxT-(T+D))/(MaxT-MinT)+(MaxT-T)/(MaxT-MinT))/2.
compute_temporal_pref2(T,D,_MinT,_MaxT,2,Point,Util):-
    T#<Point => P1#=Point-T,
    T#>=Point => P1#=0,
    (T+D)#>=Point => Util$=1.01*(1-(P1/D)),
    (T+D)#<Point => Util$=0.
compute_temporal_pref2(T,D,_MinT,_MaxT,-2,Point,Util):-
    ((T+D)#>Point) => P1#=T+D-Point,
    ((T+D)#=<Point) => P1#=0,
    T#<Point => Util$=1.01*(1-(P1/D)),
    T#>=Point => Util$=0.

compute_durutil(DurMin,DurMin,_Durutil,_Duration, 0):-!.
compute_durutil(DurMin,DurMax,Durutil,Duration, Util):-
    DurToMax$=DurMax-DurMin,
    DurToMax1$=min(Duration-DurMin,DurToMax),
    Util$=Durutil*(DurToMax1/DurToMax).

get_all_variables([],[]):-!.
get_all_variables([task(_,Subtasks,Flag,_)|Tasks], Variables):-
    get_all_variables2(Subtasks,Var1),
    append([Flag|Var1],Variables0,Variables),
    get_all_variables(Tasks,Variables0).
```

```
get_all_variables2([],[]):-!.
get_all_variables2([subtask(T,D,L)|Subtasks], [T,D,L|Variables]):-
    get_all_variables2(Subtasks,Variables).

post_distances([],_LocDistances):-!.
post_distances([task(_ID,Subtasks,Flag,_Duration)|Tasks],LocDistances):-
    post_distances2(Subtasks,Flag,Tasks,LocDistances),
    post_distances(Tasks,LocDistances).

post_distances2([],_Flag,_Tasks,_LocDistances):-!.
post_distances2([subtask(T,D,L)|Subtasks],Flag,Tasks,LocDistances):-
    post_distances3(T,D,L,Flag,1,Subtasks,LocDistances,0),
    post_distances4(T,D,L,Flag,Tasks,LocDistances),
    post_distances2(Subtasks,Flag,Tasks,LocDistances).

post_distances3(_T,_D,_L,_Flag,_Flag2,[],_LocDistances,_):-!.
post_distances3(T,D,L,Flag,_Flag2,[subtask(T2,D2,L2)|Subtasks],LocDistances,0):-
    loc_dist(L,L2,LocDist) infers most,
    (Flag#=1 and D#>0 and D2#>0) => (T+D+LocDist)#=<T2,
    post_distances3(T,D,L,Flag,1,Subtasks,LocDistances,0).
post_distances3(T,D,L,Flag,Flag2,[subtask(T2,D2,L2)|Subtasks],LocDistances,1):-
    loc_dist(L,L2,LocDist) infers most,
    (Flag#=1 and Flag2#=1 and D#>0 and D2#>0) => ((T+D+LocDist)#=<T2 or (T2+D2+LocDist)#=<T)
        ,
    post_distances3(T,D,L,Flag,Flag2,Subtasks,LocDistances,1).

post_distances4(_T,_D,_L,_Flag,[],_LocDistances):-!.
post_distances4(T,D,L,Flag,[task(_ID,Subtasks,Flag2,_Duration)|Tasks],LocDistances):-
    post_distances3(T,D,L,Flag,Flag2,Subtasks,LocDistances,1),
    post_distances4(T,D,L,Flag,Tasks,LocDistances).

build_distance(Loc1,Loc2,LocDistances,LocDist):-
    get_domain(Loc1,Loc1Domain),
    get_domain(Loc2,Loc2Domain),
    write(Loc1Domain),write(" "),write(Loc2Domain),nl,
    max_distance(LocDistances,MaxDist),
    LocDist#::0..MaxDist,
    build_distance2(Loc1Domain,Loc2Domain,Loc1,Loc2,LocDistances,LocDist).

build_distance2([],_Loc2Domain,_Loc1,_Loc2,_LocDistances,_LocDist):-!.
build_distance2([A|Loc1Domain],Loc2Domain,Loc1,Loc2,LocDistances,LocDist):-
    build_distance3(A,Loc2Domain,Loc1,Loc2,LocDistances,LocDist),
    build_distance2(Loc1Domain,Loc2Domain,Loc1,Loc2,LocDistances,LocDist).
build_distance2(A,Loc2Domain,Loc1,Loc2,LocDistances,LocDist):-
    build_distance3(A,Loc2Domain,Loc1,Loc2,LocDistances,LocDist).

build_distance3(_A,[],_Loc1,_Loc2,_LocDistances,_LocDist):-!.
build_distance3(A,[B|Loc2Domain],Loc1,Loc2,LocDistances,LocDist):-
    get_distance(A,B,LocDistances,loc_dist(A,B,Dist)),
    (Loc1#=A and Loc2#=B) => LocDist#=Dist,
    build_distance3(A,Loc2Domain,Loc1,Loc2,LocDistances,LocDist).
build_distance3(A,B,Loc1,Loc2,LocDistances,LocDist):-
    get_distance(A,B,LocDistances,loc_dist(A,B,Dist)),
    (Loc1#=A and Loc2#=B) => LocDist#=Dist.
```

```
309
310 get_distance(Loc1,Loc2,[],_):-write("Location␣Distance␣between␣"), write(Loc1), write("␣and␣
        "),write(Loc2), write("␣cannot␣be␣found."),nl,fail.
311 get_distance(Loc1,Loc2,[loc_dist(Loc1,Loc2,LocDist)|_LocDistances],loc_dist(Loc1,Loc2,
        LocDist)):-!.
312 get_distance(Loc1,Loc2,[_|LocDistances],loc_dist(Loc1,Loc2,LocDist)):-
313     get_distance(Loc1,Loc2,LocDistances,loc_dist(Loc1,Loc2,LocDist)).
314
315 max_distance([],0).
316 max_distance([loc_dist(_Loc1,_Loc2,LocDist)|LocDistances],Max):-
317     Max#=max(LocDist,Max0),
318     max_distance(LocDistances,Max0).
319
320 post_domains([]):-!.
321 post_domains([task(ID,Subtasks,Flag,Duration)|Tasks]):-
322     Flag#::0..1,
323     Duration#>=0,
324     Flag#=0 => Duration#=0,
325     task(id(ID),_,loc(Locs),dur(DurMin,DurMax,_,_),smin(Smin),smax(Smax),_,_,_,_,_,domain(
            Domain)),
326     process_domain(Domain,Smin,Domain0),
327     MaxSubTasks is integer(floor(DurMax/Smin)),
328     MinSubTasks is integer(ceiling(DurMin/Smax)),
329     subtask_num(ID,MinSubTasks,MaxSubTasks,NofSubTasks),
330     length(Subtasks,NofSubTasks),
331     initiate_subtasks_list(Flag,Subtasks,Locs,0,SumOfD,Smin,Smax,Domain0,Domain,-2),
332     Duration#=SumOfD,
333     Flag#=1 => Duration#>=DurMin,
334     Flag#=1 => Duration#=<DurMax,
335     post_domains(Tasks).
336
337 subtask_num(ID, _MinSubTasks, _MaxSubTasks, NofSubTasks):-
338     task(id(ID),_,_,dur(_, _, _, 0), _, _, _, _, _, _, _, _),
339     !,
340     NofSubTasks#=1.
341 subtask_num(ID, MinSubTasks, MaxSubTasks, NofSubTasks):-
342     task(id(ID),_,_,dur(_, _, _, 1), _, _, _, _, _, _, _, _),
343     NofSubTasks is integer(max(MaxSubTasks, MinSubTasks)).
344
345 initiate_subtasks_list(_Flag,[],_Locs,SumOfD,SumOfD,_Smin,_Smax,_Domain,_Dom,_PT).
346 initiate_subtasks_list(Flag,[subtask(T,D,Loc)|L],Locs,Dur,SumOfD,Smin,Smax,Domain,Dom,PT):-
347     T#::[-1|Domain],
348     D#::[0,Smin..Smax],
349     Loc#::[-1|Locs],
350     Flag#=0 => D#=0,
351     Flag#=0 => T#= -1,
352     Flag#=0 => Loc#= -1,
353     D#=0 => Loc#= -1,
354     D#>0 => Loc#> -1,
355     T#= -1 => D#=0,
356     T#> -1 => D#>0,
357     D#=0 => T#= -1,
358     D#>0 => T#> -1,
359     PT#= -1 => T#= -1,
```

```
    PT#>=0 => (T#>=PT or T#= -1),
    %PD#>=0 => D#=<PD,
    Dur1#=Dur+D,
    in_domain(Dom,T,D),
    initiate_subtasks_list(Flag,L,Locs,Dur1,SumOfD,Smin,Smax,Domain,Dom,T).

process_domain([],_,[]):-!.
process_domain([A..B|Domain],Smin,[A..B1|Domain0]):-
    B1 is B-Smin,
    process_domain(Domain,Smin,Domain0).

in_domain([],_,_):-!.
in_domain([A..B|Domain],T,D):-
    (T#>=A and T#<B) => (T+D)#=<B,
    in_domain(Domain,T,D).

post_ordering_constraints([],_):-!.
post_ordering_constraints([before(ID1,ID2)|OrderingConstraints],Tasks):-
    get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
    %last(Subtasks1, subtask(T1E,_D1E,_Loc1)),
    post_ordering_constraints2(Subtasks1,Subtasks2,Flag1,Flag2),
    post_ordering_constraints(OrderingConstraints,Tasks).

post_ordering_constraints2([],_Subtask2,_Flag1,_Flag2):-!.
post_ordering_constraints2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,Flag1,Flag2):-
    post_ordering_constraints3(T1,D1,Subtasks2,Flag1,Flag2),
    post_ordering_constraints2(Subtasks1,Subtasks2,Flag1,Flag2).

post_ordering_constraints3(_T1,_D1,[],_Flag1,_Flag2):-!.
post_ordering_constraints3(T1, D1, [subtask(T2,_D2,_L2)|Subtasks2], Flag1, Flag2):-
    (Flag1#=1 and Flag2#=1 and T1#>=0 and T2#>=0) => T1#=<T2,
    post_ordering_constraints3(T1,D1,Subtasks2,Flag1,Flag2).

post_mindist_constraints([],_):-!.
post_mindist_constraints([min_dist(ID1,ID2,Dist)|MinDistConstraints],Tasks):-
    get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
    post_mindist_constraints2(Subtasks1,Subtasks2,Flag1,Flag2,Dist),
    post_mindist_constraints(MinDistConstraints,Tasks).

post_mindist_constraints2([],_Subtask2,_Flag1,_Flag2,_Dist):-!.
post_mindist_constraints2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,Flag1,Flag2,Dist):-
    post_mindist_constraints3(T1,D1,Subtasks2,Flag1,Flag2,Dist),
    post_mindist_constraints2(Subtasks1,Subtasks2,Flag1,Flag2,Dist).

post_mindist_constraints3(_T1,_D1,[],_Flag1,_Flag2,_Dist):-!.
post_mindist_constraints3(T1, D1, [subtask(T2,D2,_L2)|Subtasks2], Flag1, Flag2,Dist):-
    (Flag1#=1 and Flag2#=1 and T1#>=0 and T2#>=0) => (T1+D1+Dist#=<T2 or T2+D2+Dist#=<T1),
    post_mindist_constraints3(T1,D1,Subtasks2,Flag1,Flag2,Dist).

post_maxdist_constraints([],_):-!.
post_maxdist_constraints([max_dist(ID1,ID2,Dist)|MaxDistConstraints],Tasks):-
    get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
```

```
    get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
    post_maxdist_constraints2(Subtasks1,Subtasks2,Flag1,Flag2,Dist),
    post_maxdist_constraints(MaxDistConstraints,Tasks).

post_maxdist_constraints2([],_Subtask2,_Flag1,_Flag2,_Dist):-!.
post_maxdist_constraints2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,Flag1,Flag2,Dist):-
    post_maxdist_constraints3(T1,D1,Subtasks2,Flag1,Flag2,Dist),
    post_maxdist_constraints2(Subtasks1,Subtasks2,Flag1,Flag2,Dist).

post_maxdist_constraints3(_T1,_D1,[],_Flag1,_Flag2,_Dist):-!.
post_maxdist_constraints3(T1, D1, [subtask(T2,D2,_L2)|Subtasks2], Flag1, Flag2,Dist):-
    (Flag1#=1 and Flag2#=1 and T1#>=0 and T2#>=0) => T1+Dist#>=T2+D2,
    (Flag1#=1 and Flag2#=1 and T1#>=0 and T2#>=0) => T2+Dist#>=T1+D1,
    post_maxdist_constraints3(T1,D1,Subtasks2,Flag1,Flag2,Dist).

post_implication_constraints([],_):-!.
post_implication_constraints([implies(ID1,ID2)|ImplicationConstraints],Tasks):-
    get_task(ID1,Tasks,task(ID1,_Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,_Subtasks2,Flag2,_Duration2)),
    Flag1#=1 => Flag2#=1,
    post_implication_constraints(ImplicationConstraints,Tasks).

compute_ordering_preferences([],_,0):-!.
compute_ordering_preferences([before(ID1,ID2,PrefUtil)|OrderingPreferences],Tasks,Util):-
    get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
    compute_ordering_preferences2(Subtasks1,Subtasks2,Util1,Tot),
    (Flag1#=1 and Flag2#=1) => Util$=(PrefUtil*Util1/Tot)+Util2,
    (Flag1#=0 or Flag2#=0) => Util$=Util2,
    compute_ordering_preferences(OrderingPreferences,Tasks,Util2).

compute_ordering_preferences2([],_Subtask2,0,0):-!.
compute_ordering_preferences2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,Util,Tot):-
    compute_ordering_preferences3(T1,D1,Subtasks2,Util1,Tot1),
    Util$=Util1+Util2,
    Tot#=Tot1+Tot2,
    compute_ordering_preferences2(Subtasks1,Subtasks2,Util2,Tot2).

compute_ordering_preferences3(_T1,_D1,[],0,0):-!.
compute_ordering_preferences3(T1, D1, [subtask(T2,D2,_L2)|Subtasks2],Util,Tot):-
    (D1#>0 and D2#>0) => DT#=T1+D1-T2,
    (D1#>0 and D2#>0 and (T1+D1)#=<T2) => Util$=Util2+D1*D2,
    (D1#=0 or D2#=0 or T1#>=(T2+D2)) => Util$=Util2,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and D2#=<(D2-(DT-D1))) => D2ST$=D2,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and D2#>(D2-(DT-D1))) => D2ST$=D2-(DT-
        D1),
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and ((D1-DT)*D2)#>0) => TMAX$=(D1-DT)*
        D2,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and ((D1-DT)*D2)#=<0) => TMAX$=0,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and (D1-DT)#>0 and (D2-DT)#>0) => Util
        $=Util2+TMAX+D2*(D2-1)/2-(D2-DT)*(D2-DT-1)/2,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and (D1-DT)#>0 and (D2-DT)#=<0) => Util
        $=Util2+TMAX+D2*(D2-1)/2,
```

```
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and (D1-DT)#=<0 and D2ST#>D1) => Util$=Util2-TMAX+D2ST*(D2ST-1)/2-(D2ST-D1)*(D2ST-D1-1)/2,
    (D1#>0 and D2#>0 and (T2+D2)#>T1 and (T1+D1)#>T2 and (D1-DT)=<0 and D2ST=<D1) => Util$=Util2+TMAX+D2ST*(D2ST-1)/2,
    Tot#=Tot1+D1*D2,
    compute_ordering_preferences3(T1,D1,Subtasks2,Util2,Tot1).

compute_mindist_preferences([],_,0):-!.
compute_mindist_preferences([min_dist(ID1,ID2,Dist,PrefUtil)|MinDistPreferences],Tasks,Util):-
    get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
    compute_mindist_preferences2(Subtasks1,Subtasks2,Dist,Util1,Tot),
    (Flag1#=1 and Flag2#=1) => Util$=PrefUtil*(Util1/Tot)+Util2,
    (Flag1#=0 or Flag2#=0) => Util$=Util2,
    compute_mindist_preferences(MinDistPreferences,Tasks,Util2).

compute_mindist_preferences2([],_Subtask2,_MinDist,0,0):-!.
compute_mindist_preferences2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,MinDist,Util,Tot):-
    compute_mindist_preferences3(T1,D1,Subtasks2,MinDist,Util1,Tot1),
    Util$=Util1+Util2,
    Tot#=Tot1+Tot2,
    compute_mindist_preferences2(Subtasks1,Subtasks2,MinDist,Util2,Tot2).

compute_mindist_preferences3(_T1,_D1,[],_MinDist,0,0):-!.
compute_mindist_preferences3(T1, D1, [subtask(T2,D2,_L2)|Subtasks2],MinDist,Util,Tot):-
    (D1#>0 and D2#>0 and T2#>=T1) => D#=MinDist-(T2-(T1+D1)),
    (D1#>0 and D2#>0 and T1#>T2) => D#=MinDist-(T1-(T2+D2)),
    (D1#>0 and D2#>0 and MinDist#=<(T2-(T1+D1)) and T2#>=T1) => Util$=Util2+D1*D2,
    (D1#=0 or D2#=0 or (MinDist#>=(T2+D2-T1) and T2#>=T1)) => Util$=Util2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and D2#=<(D2-(D-D1))) => D2S$=D2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and D2#>(D2-(D-D1))) => D2S$=D2-(D-D1),
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and ((D1-D)*D2)#>0) => TMIN$=(D1-D)*D2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and ((D1-D)*D2)#=<0) => TMIN$=0,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and (D1-D)#>0 and (D2-D)#>0) => Util$=Util2+TMIN+D2*(D2-1)/2-(D2-D)*(D2-D-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and (D1-D)#>0 and (D2-D)#=<0) => Util$=Util2+TMIN+D2*(D2-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and (D1-D)#=<0 and D2S#>D1) => Util$=Util2+TMIN+D2S*(D2S-1)/2-(D2S-D1)*(D2S-D1-1)/2,
    (D1#>0 and D2#>0 and (T2+D2-T1)#>MinDist and (T2-(T1+D1))#<MinDist and T2#>=T1 and (D1-D)#=<0 and D2S#=<D1) => Util$=Util2+TMIN+D2S*(D2S-1)/2,
    (D1#>0 and D2#>0 and MinDist#=<(T1-(T2+D2)) and T2#<T1) => Util$=Util2+D1*D2,
    (D1#=0 or D2#=0 or (MinDist#>=(T1+D1-T2) and T2#<T1)) => Util$=Util2,
    (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and D1#=<(D1-(D-D2))) => D1S$=D1,
    (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and D1#>(D1-(D-D2))) => D1S$=D1-(D-D2),
    (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and ((D2-D)*D1)#>0) => TMIN$=(D2-D)*D1,
```

```
503     (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and ((D2-D
            )*D1)#=<0) => TMIN$=0,
504     (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and (D2-D)
            #>0 and (D1-D)#>0) => Util$=Util2+TMIN+D1*(D1-1)/2-(D1-D)*(D1-D-1)/2,
505     (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and (D2-D)
            #>0 and (D1-D)#=<0) => Util$=Util2+TMIN+D1*(D1-1)/2,
506     (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and (D2-D)
            #=<0 and D1S#>D2) => Util$=Util2+TMIN+D1S*(D1S-1)/2-(D1S-D2)*(D1S-D2-1)/2,
507     (D1#>0 and D2#>0 and (T1+D1-T2)#>MinDist and (T1-(T2+D2))#<MinDist and T2#<T1 and (D2-D)
            #=<0 and D1S#=<D2) => Util$=Util2+TMIN+D1S*(D1S-1)/2,
508     Tot#=Tot1+D1*D2,
509     compute_mindist_preferences3(T1,D1,Subtasks2,MinDist,Util2,Tot1).
510
511 compute_maxdist_preferences([],_,0):-!.
512 compute_maxdist_preferences([max_dist(ID1,ID2,Dist,PrefUtil)|MaxDistPreferences],Tasks,Util)
        :-
513     get_task(ID1,Tasks,task(ID1,Subtasks1,Flag1,_Duration1)),
514     get_task(ID2,Tasks,task(ID2,Subtasks2,Flag2,_Duration2)),
515     compute_maxdist_preferences2(Subtasks1,Subtasks2,Dist,Util1,Tot),
516     (Flag1#=1 and Flag2#=1) => Util$=PrefUtil*(Util1/Tot)+Util2,
517     (Flag1#=0 or Flag2#=0) => Util$=Util2,
518     compute_maxdist_preferences(MaxDistPreferences,Tasks,Util2).
519
520 compute_maxdist_preferences2([],_Subtask2,_MaxDist,0,0):-!.
521 compute_maxdist_preferences2([subtask(T1,D1,_L1)|Subtasks1],Subtasks2,MaxDist,Util,Tot):-
522     compute_maxdist_preferences3(T1,D1,Subtasks2,MaxDist,Util1,Tot1),
523     Util$=Util1+Util2,
524     Tot#=Tot1+Tot2,
525     compute_maxdist_preferences2(Subtasks1,Subtasks2,MaxDist,Util2,Tot2).
526
527 compute_maxdist_preferences3(_T1,_D1,[],_MaxDist,0,0):-!.
528 compute_maxdist_preferences3(T1, D1, [subtask(T2,D2,_L2)|Subtasks2],MaxDist,Util,Tot):-
529     (D1#>0 and D2#>0 and T2#>=T1) => DT#=T2+D2-T1-MaxDist,
530     (D1#>0 and D2#>0 and T1#>T2) => DT#=T1+D1-T2-MaxDist,
531     (D1#>0 and D2#>0 and MaxDist#>=(T2+D2-T1) and T2#>=T1) => Util$=Util2+D1*D2,
532     (D1#=0 or D2#=0 or (MaxDist#=<(T2-(T1+D1)) and T2#>=T1)) => Util$=Util2,
533     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and D2#=<
            (D2-(DT-D1))) => D2ST$=D2,
534     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and D2#>(
            D2-(DT-D1))) => D2ST$=D2-(DT-D1),
535     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and ((D1-
            DT)*D2)#>0) => TMAX$=(D1-DT)*D2,
536     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and ((D1-
            DT)*D2)#=<0) => TMAX$=0,
537     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and (D1-
            DT)#>0 and (D2-DT)#>0) => Util$=Util2+TMAX+D2*(D2-1)/2-(D2-DT)*(D2-DT-1)/2,
538     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and (D1-
            DT)#>0 and (D2-DT)#=<0) => Util$=Util2+TMAX+D2*(D2-1)/2,
539     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and (D1-
            DT)#=<0 and D2ST#>D1) => Util$=Util2+TMAX+D2ST*(D2ST-1)/2-(D2ST-D1)*(D2ST-D1-1)/2,
540     (D1#>0 and D2#>0 and (T2-(T1+D1))#<MaxDist and (T2+D2-T1)#>MaxDist and T2#>=T1 and (D1-
            DT)=<0 and D2ST=<D1) => Util$=Util2+TMAX+D2ST*(D2ST-1)/2,
541     (D1#>0 and D2#>0 and MaxDist#>=(T1+D1-T2) and T2#<T1) => Util$=Util2+D1*D2,
542     (D1#=0 or D2#=0 or (MaxDist#=<(T1-(T2+D2)) and T2#<T1)) => Util$=Util2,
```

```
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and D1#=<(D1-(DT-D2))) => D1ST$=D1,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and D1#>(D1-(DT-D2))) => D1ST$=D1-(DT-D2),
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and ((D2-DT)*D1)#>0) => TMAX$=(D2-DT)*D1,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and ((D2-DT)*D1)#=<0) => TMAX$=0,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and (D2-DT)#>0 and (D1-DT)#>0) => Util$=Util2+TMAX+D1*(D1-1)/2-(D1-DT)*(D1-DT-1)/2,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and (D2-DT)#>0 and (D1-DT)#=<0) => Util$=Util2+TMAX+D1*(D1-1)/2,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and (D2-DT)#=<0 and D1ST#>D2) => Util$=Util2+TMAX+D1ST*(D1ST-1)/2-(D1ST-D2)*(D1ST-D2-1)/2,
    (D1#>0 and D2#>0 and (T1-(T2+D2))#<MaxDist and (T1+D1-T2)#>MaxDist and T2#<T1 and (D2-DT)=<0 and D1ST=<D2) => Util$=Util2+TMAX+D1ST*(D1ST-1)/2,
    Tot#=Tot1+D1*D2,
    compute_mindist_preferences3(T1,D1,Subtasks2,MaxDist,Util2,Tot1).

compute_implication_preferences([],_,0):-!.
compute_implication_preferences([implies(ID1,ID2,PrefUtil)|ImplicationPreferences],Tasks,Util):-
    get_task(ID1,Tasks,task(ID1,_Subtasks1,Flag1,_Duration1)),
    get_task(ID2,Tasks,task(ID2,_Subtasks2,Flag2,_Duration2)),
    (Flag1#=1 and Flag2#=1) => Util$=Util1+PrefUtil,
    (Flag1#=0 or Flag2#=0) => Util$=Util1,
    compute_implication_preferences(ImplicationPreferences,Tasks,Util1).

get_task(ID,[],_):-write("Task␣"), write(ID), write("␣cannot␣be␣found."),nl,fail.
get_task(ID,[task(ID,Subtasks,Flag,Duration)|_Tasks],task(ID,Subtasks,Flag,Duration)):-!.
get_task(ID,[_|Tasks],task(ID,Subtasks,Flag,Duration)):-
    get_task(ID,Tasks,task(ID,Subtasks,Flag,Duration)).
```

APPENDIX D

Problem Definition of "A Realistic Scenario"

The following code is the definition of the problem instance "A Realistic Scenario", from Chapter 3, as it was given to the two schedulers. It serves as an example of the format of problem instances that the scheduler recognizes. The following code was automatically generated by SelfPlanner, through the use of its API; Ann's problem was defined through the Object-Oriented higher-level model of SelfPlanner.[1]

Listing D.1: ann-problem.ecl

```
nof_tasks(18).
task(id(0), task_locs(1), loc([1]), dur(1,1,0.000000,0), smin(1), smax(1), prox_cons
    (0,9999999), prox_prefs(0,0.000000,9999999,0.000000), utility(1000.000000), utilization
    (1.000000), domain([[0..1,0.0,0.0]])).
task(id(1), task_locs(1), loc([1]), dur(4,12,80.000000,1), smin(2), smax(4),prox_cons
    (1,9999999), prox_prefs(0,0.000000,9999999,0.000000), utility(30.000000), utilization
    (1.000000), domain([[2..21,0.7895,15.00], [49..69,0.7895,15.00], [97..117,0.7895,15.00],
     [145..165,0.7895,15.00]])).
task(id(2), task_locs(1), loc([1]), dur(4,12,80.000000,1), smin(2), smax(4), prox_cons
    (1,9999999), prox_prefs(0,0.000000,9999999,0.000000), utility(30.000000), utilization
    (1.000000), domain([[2..21,0.7895,15.00], [49..69,0.7895,15.00], [97..117,0.7895,15.00],
     [145..165,0.7895,15.00]])).
task(id(3), task_locs(1), loc([1]), dur(6,10,40.000000,1), smin(2), smax(4), prox_cons
    (1,9999999), prox_prefs(0,0.000000,9999999,0.000000), utility(40.000000), utilization
    (1.000000), domain([[2..21,15.00,0.7895], [49..69,15.00,0.7895], [97..117,15.00,0.7895],
     [145..165,15.00,0.7895]])).
task(id(4), task_locs(1), loc([3]), dur(2,4,30.000000,0), smin(2), smax(4), prox_cons
    (1,9999999), prox_prefs(0,0.000000,9999999,0.000000), utility(30.000000), utilization
    (1.000000), domain([[17..21,1,1], [55..61,1,1]])).
task(id(5), task_locs(1), loc([1]), dur(6,12,100.000000,1), smin(3), smax(6), prox_cons
    (1,9999999), prox_prefs(0,0.000000,48,20.000000), utility(80.000000), utilization
    (1.000000), domain([[2..21,0.7895,15.00], [49..69,0.7895,15.00], [97..117,0.7895,15.00],
     [145..165,0.7895,15.00]])).
```

[1]Notice that SelfPlanner automatically inserts task 0 into the generated problem instance with the user's current location so that the scheduler takes it into account. This task should be ignored from the solution and its utility subtracted (−1000 utility).

```
8  task(id(6), task_locs(1), loc([1]), dur(4,4,0.00,0), smin(4), smax(4), prox_cons(1,9999999),
       prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000), domain
       ([[3..7,1,1]])).
9  task(id(7), task_locs(1), loc([1]), dur(4,4,0.00,0), smin(4), smax(4), prox_cons(1,9999999),
       prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000), domain
       ([[15..19,1,1]])).
10 task(id(8), task_locs(1), loc([1]), dur(4,4,0.00,0), smin(4), smax(4), prox_cons(1,9999999),
       prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000), domain
       ([[59..63,1,1]])).
11 task(id(9), task_locs(1), loc([1]), dur(4,4,0.00,0), smin(4), smax(4), prox_cons(1,9999999),
       prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000), domain
       ([[97..101,1,1]])).
12 task(id(10), task_locs(1), loc([1]), dur(4,4,0.00,0), smin(4), smax(4), prox_cons(1,9999999)
       , prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000),
       domain([[147..151,1,1]])).
13 task(id(11), task_locs(1), loc([1]), dur(2,2,0.00,0), smin(2), smax(2), prox_cons(1,9999999)
       , prox_prefs(0,0.00,99999999,0.000000), utility(40.000000), utilization(1.000000),
       domain([[153..163,1,1]])).
14 task(id(12), task_locs(1), loc([1]), dur(5,5,0.00,0), smin(5), smax(5), prox_cons(1,9999999)
       , prox_prefs(0,0.00,99999999,0.000000), utility(80.000000), utilization(1.000000),
       domain([[107..112,1,1]])).
15 task(id(13), task_locs(1), loc([1]), dur(4,6,20.00,0), smin(4), smax(6), prox_cons
       (1,9999999), prox_prefs(0,0.00,99999999,0.000000), utility(60.000000), utilization
       (1.000000), domain([[49..57,1,1]])).
16 task(id(14), task_locs(1), loc([1]), dur(1,2,20.00,0), smin(1), smax(2), prox_cons
       (1,9999999), prox_prefs(0,0.00,99999999,0.000000), utility(60.000000), utilization
       (1.000000), domain([[13..17,5.00,0.00]])).
17 task(id(15), task_locs(1), loc([1]), dur(1,2,20.00,0), smin(1), smax(2), prox_cons
       (1,9999999), prox_prefs(0,0.00,99999999,0.000000), utility(60.000000), utilization
       (1.000000), domain([[61..65,5.00,0.00]])).
18 task(id(16), task_locs(1), loc([1]), dur(1,2,20.00,0), smin(1), smax(2), prox_cons
       (1,9999999), prox_prefs(0,0.00,99999999,0.000000), utility(60.000000), utilization
       (1.000000), domain([[109..113,5.00,0.00]])).
19 task(id(17), task_locs(1), loc([1]), dur(1,2,20.00,0), smin(1), smax(2), prox_cons
       (1,9999999), prox_prefs(0,0.00,99999999,0.000000), utility(60.000000), utilization
       (1.000000), domain([[157..161,5.00,0.00]])).
20 nof_locations(4).
21 loc_dist(0,0,0).
22 loc_dist(0,1,0).
23 loc_dist(0,2,0).
24 loc_dist(0,3,0).
25 loc_dist(1,0,0).
26 loc_dist(1,1,0).
27 loc_dist(1,2,2).
28 loc_dist(1,3,2).
29 loc_dist(2,0,0).
30 loc_dist(2,1,2).
31 loc_dist(2,2,0).
32 loc_dist(2,3,4).
33 loc_dist(3,0,0).
34 loc_dist(3,1,2).
35 loc_dist(3,2,4).
36 loc_dist(3,3,0).
37 ordering_constraints(1).
```

```
before(4,5).
min_dist_constraints(0).
max_dist_constraints(0).
implication_constraints(2).
implies(5,4).
implies(4,5).
ordering_preferences(0).
min_dist_preferences(0).
max_dist_preferences(0).
implication_preferences(0).
```

APPENDIX E

Scheduler Options

SWO, the scheduler for the problem formulated in Section 1.1, was extended with the support of (a) Non-monotonic temporal preferences, (b) Traveling Time Minimization, (c) the set of transformations of valid plans, (d) local-optimization post-processing (using hill-climbing), (e) The modified Simulated Annealing empowered with Tabu Lists and backtracking algorithm, (f) PDiff function, (g) weights on utility sources to implement and evaluate the on-line learning of preferences method, (h) a command-line options parser, (i) many optimizations to the codebase. The codebase is in C++.

The default options are for the scheduler to produce one plan using SWO+SA2K. Some important parameters are: `-k` to change the KMAX parameter of the SA algorithm, `-o` to use SA as the main scheduler and not run SWO, `-l` to local-optimize the final solution and `-r` to produce multiple plans. An example of the problem instance format the scheduler recognizes is included in Appendix D. What follows is a list of all the scheduler parameters:

```
$ swo -h
swo version 2.99
Usage: swo <parameters> filename.ecl
-h This help message
-v Print version number and exit
-o Do not use SWO. Use SA as the main scheduler
-k <iterations> Number of iterations. Default is 2000. Set to 0 to disable post-
processing
-l Add local optimization at the end
-b <0/1> Weighted Backtracking off/on. Default is on
-f <1/2/3> Plan Diff function used for evaluating alternative plans. Default is
3(slowest), 2 is faster
-w Plan_Diff_Dist_Weight Plan_Diff_Dur_Weight Plan_Diff_Loc_Weight
Plan_Diff_Pairs_Weight. Default is 0.25 0.25 0.25 0.25
```

```
   -r <number_of_runs> Number of plans produced. Maximizes difference between
solutions
   -i <0/1> Recompute NM_TIMELINE parameter for new U(Time) function
   -d Disable Location Changes
   -c Use Complete Add Task transformation (slower)
   -p <par> Plan Difference Multiplier. Default is 1.0
   -t <par> Initial Temprature (0..1). Default is 0.9
   -n <par> Cooling Schedule Difference parameter. Default is 0.07
   -e <par> Stage Thresh parameter (0..1) default is 1.0
   -g <nof> Generate a problem with <nof> number of tasks and exit
   -s <par> Minimum Plan Tightness (0..1) to enable Trim Activity Parts Trans-
formation before post-processing. Default is 0.9. Set to > 1 to disable
   -a <crit_w0 crit_w1 ... crit_w7 crit_w8> Learning weights for the nine crite-
ria. Default is 1
   -8 <user_pref0 user_pref1 ... user_pref7 user_pref8> Users preference parame-
ters. Default is 1
   -q <0/1> Enable/Disable sorting. Default is 1
   -j <distance utility> Enable traveling distance minimization. Provide utility
for max free time. Default is off
   -x Run in DEBUG mode
```

Index