Semester Project in Computer Science

Multi-agent Meeting Scheduling with Privacy Guarantees

Achraf Tanguy
EPFL
School of Computer and Communication Sciences
Spring 2010

**Professor:** Boi Faltings

**Supervisors: Artificial Intelligence Lab**
Thomas Léauté
Florent Garcin
Radoslaw Szymanek

**Supervisors: Nokia Research Center**
Gian Paolo Perrucci
Imad Aad
Contents

1 Introduction 3

2 Related Work 3
   2.1 Distributed Multi-event Scheduling 3
   2.2 Solving DCOP problems 4
      2.2.1 ADOPT- Asynchronous Distributed Optimization 4
      2.2.2 SynchBB- Synchronous branch and bound 4
      2.2.3 DPOP- Distributed Pseudo-tree Optimisation Procedure 4
   2.3 Privacy issues in DCOPs 5
      2.3.1 Privacy guarantee types 5
      2.3.2 P-DPOP: solving DCOP with privacy guarantees 6
      2.3.3 P^2-DPOP: 6
   2.4 Studies in the field of user preferences: 6
      2.4.1 CMRadar: A Personal Assistant Agent for Calendar Management 7
      2.4.2 Optimizing agent-based meeting scheduling through preference estimation 7
      2.4.3 Learning other’s calendar 8

3 NRC Lausanne Calendar DATABASE 9
   3.1 Data Structure 9
      3.1.1 Description 9
      3.1.2 Some Definitions 10
      3.1.3 Performed work and encountered limitations 11
   3.2 Results 12
      3.2.1 Phone calendar usage 12
      3.2.2 Appointment scheduling 13
   3.3 Discussions 14

4 The problem generator 14
   4.1 Arbitrary, Satisfaction Problems 14
   4.2 Real-life-looking Satisfaction Problems 14
   4.3 Real-life-looking Optimisation Problems 15

5 Experimental results 15
   5.1 Effect of the number of agents 15
   5.2 Effect of the number of meetings 16
   5.3 Effect of the number of agents per meeting 16
   5.4 Effect of preexisting meetings 16
   5.5 Effect of adding preferences 17

6 Conclusion and future work 18
Abstract

In this paper we present benchmarks for some of the state-of-the-art algorithms to solve the DCOP family of problems. We especially focus on meeting scheduling, and for this goal we present an open-source random problem generator that produces DCOP instances having some of the characteristics of real-life scheduling problems, such as preferences and preexisting meetings. Real-life-looking problems were inferred by data-mining an existing database provided by the Nokia Research Center in Lausanne.

1 Introduction

The Multi-agents meeting scheduling problems (MSP) are the set of problems that deal with solving situations where different agents have a certain number of meetings, either independent or interdependent with each others, and each of those meetings has to be assigned a slot that is not overlapping with others.

An MSP can be formulated as a mathematical problem using DiMES. This obtained problem belongs to the so called Distributed Constraint Optimization Problems (DCOPs) [1], the former have been studied in several research projects.

In this scope the Artificial Intelligence Laboratory (LIA) at EPFL developed an algorithm, DPOP, that solves DCOPs using dynamic programming [2]. Furthermore, this algorithm was improved by adding some privacy guarantees which yielded to P-DPOP [3].

DPOP has been tested for real-life meeting scheduling showing better results than other algorithms [2]. However no experiments have been made for P-DPOP. The goal of this project is first to implement a parametrized problem generator (PPG) that outputs meeting scheduling problems, which will serve as test base for the DPOP and P-DPOP algorithms. The tests show how those algorithms perform compared to simple search algorithm such as ADOPT [4] or SynchBB [5].

In a further step the PPG will be improved by outputting real-life-looking problems. This is possible through the data-mining that we will be doing on the database provided by the Nokia Research Center in Lausanne.

2 Related Work

2.1 Distributed Multi-event Scheduling

DiMES (Distributed Multi-Event Scheduling) is a framework that has been introduced with the aim of capturing a rich class of real-world problems involving joint activities [6], among the class of problems that fall into the framework’s scope we find meeting scheduling. The properties of a DiMES are the following:

- A resources set \( R := \{R_1, \ldots, R_n\} \) of cardinality \( N \) where \( R_n \) refers to the \( n \)-th resource. In the case of meeting scheduling, resources include participants, meeting rooms...
- An events set \( E := \{1, \ldots, E^K\} \) of cardinality \( K \) where \( E^K \) refers to the \( k \)-th event.
- A time domain \( \Gamma := \{1, \ldots, T\} \) of cardinality \( T \), where the element \( t \in \Gamma \) refers to the beginning of a time slot.
- Formulation:

\[
E^K := (A^K, L^K, V^K)
\]

\( A^K \) : the set of resources needed by \( E^K \)

\( L^K \) : length of event

\( V^K \) : valuation of event \( k \) for time slot

According to the DiMES framework, scheduling problems can be modelled as:

A schedule \( S \) as a mapping from the event set to the time domain where \( S(E^K) \in \Gamma \) denotes the time slots committed for event \( k \). An event is not disjoint, i.e. event \( E^K \) must be scheduled in \( L^K \) contiguous slots.

The utility function is defined as follows:

\[
(x_1^*, \ldots, x_n^*) = \arg \max_{(x_1, \ldots, x_n)} \sum_i c_i.
\]

where \( c_i = \text{reward} - V^K_i \).
DCOP Formulations for DiMES  Now that we have captured our problem in the DiMES framework, we need an approach to find an (optimal) solution.

A DCOP is defined as a tuple \( < A, X, D, R > \) where \( A \) is a finite set of agents \( A = \{ A_1, ..., A_m \} \), \( X \) is a set of variables \( X = \{ x_1, ..., x_N \} \) distributed among the agents in \( A \) (each variable belongs to a unique agent), \( D \) is a set of domains \( D = \{ D_1, ..., D_N \} \) where a domain \( D_i \) is the set of values that variable \( X_i \) can take. Finally \( R \) is a set of relations \( R = \{ r_1, ..., r_p \} \) such that every relation defines a cost for the combinations of values from \( D_i \).

The goal is to choose values for variables to optimize an objective function, the utility. The conversion from a DiMES to a DCOP can be made in 3 different ways [6]:

- **TSA V (Time Slots As Variables)**: a variable \( X_n(t) \) is the \( n \)-th resource’s \( t \)-th time slot.
  ⇒ The complexity of TSAV grows if the time range increases or the time quantization interval decreases.

- **EAV (Event As Variables)**: \( X^K \) represents the starting time of \( E^K \), with domain \( \{ t, ..., t + L^K - 1 \} \).

- **PEAV (Private Events As Variables)**: A modification of EAV to ensure privacy protection for agents’ information. As in EAV each participant controls a private variable corresponding to the starting time of a meeting. But the difference is that in this formulation agents don’t share their internal valuation for the resources, thus maintaining privacy. The experimental results in [6] shows that EAV outperforms PEAV by one order of magnitude when using passup in meeting scheduling. As in this project it is fundamental to preserve agents preferences as well as to have the best variable control scheme we will consider only PEAV formulation.

2.2 Solving DCOP problems

There are many algorithms that solve DCOP problems. These algorithms have different characteristics; we present in this section three algorithms, together with a pros and cons analysis.

2.2.1 ADOPT- Asynchronous Distributed Optimization

ADOPT is claimed to be the first algorithm to be distributed, asynchronous and optimal for DCOP as well as being efficient and polynomial space for each of the agents [4]. The algorithm is built on top of the backtracking method. More specifically it uses the weak notion of this search method that is to use a lower bound when looking for solutions, and search in parallel in independent sub-trees. ADOPT uses then efficient reconstruction of abandoned solutions to control the optimality of the backtrack solution. The main drawback of ADOPT is that it has a worst-case complexity (in terms of number of messages exchanged) that is exponential in the depth of the pseudo-tree arrangement [2]. This is due to the fact that all the variables of the longest path root → leaf have to be instantiated and their different values tried out.

2.2.2 SynchBB- Synchronous branch and bound

Synchronous Branch and Bound is an algorithm that solves DCOPs [5], the algorithm works as follows: at each step an agent tries to assign a value to the current assignment with the condition that the lower and upper bound should not reach each other. One of the drawbacks of the algorithm is that it requires agents to perform computation in a sequential manner in which only one agent executes at a time. Agent’s order of execution is determined by a priority.

2.2.3 DPOP- Distributed Pseudo-tree Optimisation Procedure

Taking a different approach to the problem, DPOP uses dynamic programming which proves to be a very efficient method for multi-agent optimisation, because it involves sending only a small number of (large) messages, instead of a large number of small messages. DPOP is formulated for both satisfaction and optimisation problem. In the following we give an overview of the algorithm’s steps and its complexity. DPOP’s 3 phases:

- **Pseudotree construction**  In this phase a pseudo-tree, the Depth First Search tree, is generated from the graph of the DCOP. The construction of the tree is made using a leader election followed by top down pseudo-tree creation via message exchange.

- **UTIL propagation** This is the second step of the algorithm, it is a bottom up stream where nodes, starting from the leaves, send to their parents a UTIL message that summarises the maximal utility achievable by their sub-trees, as functions of decisions higher up in the DFS, as shown in Figure 1.

  Table 1 shows an example of utility message, in this case the message is sent form node \( x \) to node \( w \) in Figure 1.
Figure 1: DFS tree of a problem, with the UTIL messages exchanged by DPOP.

<table>
<thead>
<tr>
<th>$x \rightarrow w$</th>
<th>$w = v^0_w$</th>
<th>$w = v^1_w$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTIL</td>
<td>$u^*_x(v^0_w)$</td>
<td>$u^*_x(v^1_w)$</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Example of UTIL message sent from node $x$ to node $w$

**VALUE propagation**  This step is mainly about informing the nodes about the chosen solution, the propagation is top down, each node receives from its parent a value message that holds the vector of values decided upon, and the receiving node sets its value accordingly.

**DPOP’s complexity**  The number of messages exchanged during the UTIL and VALUE phases is linear: $n - 1$ UTIL messages, and $m$ linear size value messages ($m$ is the number of edges). During the DFS construction phase, a quadratic number of (small) messages are exchanged. The complexity of the algorithm lies in the size of UTIL messages.

It has been proved that the largest UTIL message produced by DPOP is space exponential in the width of the pseudotree induced by the DFS ordering used [2]. This means that the choice of the pseudotree affects the efficiency of the algorithm.

### 2.3 Privacy issues in DCOPs

When looking at ways to solve DCOP problems, most existing algorithms make the assumption that agents are willing to give all the necessary information in order to solve the scheduling problem. This assumption may not always be true since some agents would rather disclose minimal information and still obtain a solution for the distributed constraints problem. The set of algorithms that provide this kind of functionality are said to have privacy guarantees.

#### 2.3.1 Privacy guarantee types

Before going more in details about the various types of privacy, let us first define *semi-private information*, a very important notion in the field of privacy preservation. Semi-private information is the information that can be inferred from the potential solutions. In distributed constraints problems, participants must accept to leak the information associated with all the possible solutions that may be reached in the end.

One can identify four types of privacy [3]:

- **Agent Privacy**: no agent can learn the identity of any other agent unless they share some constraint.

- **Topology Privacy**: no agent can learn about the topological constructs (constraints, cycles), apart from those that involve one of its variables.

- **Constraints Privacy**: no agent can learn about the content or the nature of the constraints that do not involve its variables.
• **Decision Privacy:** no agent can discover the decision that other agents make in the final solution.

![Four types of privacy guarantees](image)

**Figure 2: Four types of privacy guarantees**

### 2.3.2 P-DPOP: solving DCOP with privacy guarantees

In DPOP, privacy is lost during the 3 steps of the algorithm (i.e. DFS construction, UTIL propagation, and VALUE propagation). Moreover each of those steps allows agents to learn one or more of the 4 privacy dimensions (agent, topology, constraints, and decision)[3]. P-DPOP addresses the privacy issues mentioned above, by taking a set of measures that protect privacy at DFS tree construction, UTIL and VALUE propagation stages. Here is an overview of the measures taken by P-DPOP[3]:

- **DFS Construction:** During this first phase, the algorithm uses both Anonymous Leader Election to determine the root variable, and an anonymized form of Distributed DFS Traversal for the actual DFS construction.
- **UTIL Propagation:** In this phase the algorithm protects constraint privacy by obfuscating the constraint values by adding large random numbers.
- **VALUE Propagation:** The last phase of the algorithm uses codenames to protect decision privacy. This is however only a limited protection.

![Performance in guarantee preservation](image)

**Figure 3: P-DPOP’s performance in guarantee preservation**

Combining all those techniques, P-DPOP achieves total agent and topology privacy, but only limited constraint and decision privacy. This is due to the fact the algorithm leaks agents’ decisions to other agents that are involved in common constraints, and the obfuscation is not cryptographically secure.

**P-DPOP’s complexity**  The gain of privacy has a price, and this price is the complexity of the algorithm that increases due to some of the measures taken to avoid privacy loss. Building DFS trees in an anonymous way leads in the average to trees with bigger induced width. As stated for DPOP complexity, the complexity in size of message is exponential in the induced width. In other words, the size of UTIL messages will increase exponentially because of the number of variables that constitutes the UTIL message.

### 2.3.3 $P^2$-DPOP:

This is another algorithm that provides full privacy guarantees, by using Secure Multiparty Computation based on ElGamal encryption and cooperative decryption of messages [7]. The agents in this algorithm permute the root variable and the communication structure. At every iteration, a feasible assignment is computed for the root variable, using distributed, secure multi-party computation.

### 2.4 Studies in the field of user preferences:

Different studies have been made in the field of user preferences in meeting scheduling. In the following we show the state of the art in this subject, together with some of the projects that were accomplished so far.
2.4.1 CMRadar: A Personal Assistant Agent for Calendar Management

CMRadar claims to be the first end-to-end agent for automated calendar management, it handles negotiations with other users, makes autonomous decision, and provides interfacing and visualization [8].

CMRadar uses a representation for calendars called Templates. This data structure model is used by the different components in CMRadar to communicate calendar information. Templates are also used to normalize unformatted human readable language to machine readable format [8].

One of the most interesting modules of the agent is the Scheduler. It has the role of receiving/initiating meeting requests, it is also able to return the user availability considering their preferences and other commitments. The scheduler is using a rich set of hard and soft constraints to emulate calendar management.

In the CMRadar project learning user preference was a reactive approach, meaning that each time the user interacted with the system (by setting, refusing, or accepting a meeting) the preferences were learnt. Two types of preferences were differentiated:

- Preferences inserted by the user (e.g. I prefer to meet between 02:00pm and 04:00pm)
- Preferences learned by machine learning approach, typically this involved user-agent watching user behaviours.

![Figure 4: Schema showing architecture of CMRadar](image)

In further studies [9, 10], a statistical approach was used to learn a static time-of-day (TOD) preference curve for a given user from observed meeting scheduling data.

2.4.2 Optimizing agent-based meeting scheduling through preference estimation

This project [11] aimed to develop a mobile agent that would automate the task of meeting scheduling. The software behind is called a secretary agent. It handles negotiations with other secretary agents, and has access to its users preferences and constraints.

**Modeling** In this research a meeting scheduling has a set of parameters: Initiator, rank, attendees, type, period, duration, part-of-day, and day of week. The preference vector is a sorted vector of a person’s preferences on different alternative proposals for an announced event, such as a meeting. The vector is sorted according to the value of the preference level of a proposed alternative.

Two algorithms are used

- Algorithm 1 is a negotiation process [11] that uses the “earliest” timeslot heuristics to guide the negotiation. We talk about a relaxation process.
- Algorithm 2 extends Algorithm 1 with what we call preference estimation [11] that allows an MA to “estimate” the global preference level of a proposal and then select those proposals that are more likely to get accepted, hence generating higher quality solutions.
2.4.3 Learning other’s calendar

This work presents a method for aiding the process of meeting scheduling through learning about the meetings in the calendars of other users [12]. The privacy issue is addressed differently as in this method the software agent tries to represent other agents’ calendars as a probability distribution of possible meeting types and present an algorithm (LOC) for learning those distributions, based on responses to meeting requests. For each time slot in a response the algorithm updates the beliefs about that time slot in the responder’s calendar based on the response using a Bayesian update method. This system shows to be successful in learning the meetings of others, and it learns them within just a few weeks of negotiations [12]. Once the meetings of others calendars are learned. This information is used to decrease the amount of messages exchanged. This allows giving out less information while creating less communication overhead in negotiation. This was used to improve the meeting negotiation process by decreasing the number of messages sent while scheduling meetings in slots with high utility.

This work was integrated in the CMRadar project.
3 NRC Lausanne Calendar DATABASE

Nokia Research center of Lausanne (NRC) collected some data on mobile phone usage from a study [13] that included over 180 volunteers. Participants were provided with a Nokia N95 phone (Figure 5). All the data have been conveyed from the phones to a database after anonymizing it to protect privacy of the participants.

The data-mining will be used for two different purposes:
- Results will serve as a base for advanced researches on how people use calendars on mobile phones.
- To get an incentive on the real-life meeting scheduling. This will enable us to better shape the problem generator (see Section 4).

3.1 Data Structure

The structure of the database is rather complex due to the diversity of data collected, in this paper we will only focus on the relevant tables while highlighting the important relations with other tables.

3.1.1 Description

The table “records” holds records of activities of users when interacting with different applications in their phone. Most importantly everything that happens on the phone is time stamped and kept in this table. This table is linked to almost all the other main tables making it possible to track the activity and have a better description. The table below is an overview of tuples present in the database.

| id       | userid | time         | type   |...
|----------|--------|--------------|--------|---
| 87624719 | 61     | 12712131313  | gsm    |   |
| 87624700 | 100    | 127121212    | bt     |   |

Table 2: Partial overview of “records” table

- **id** is a unique identifier to whom other tables such as calendar point to.
- **userid** is a unique identifier of the users in the campaign.
- **time** is a time stamp, in POSIX format, of some event, the event can be anything that happened in the phone for instance incoming call or opening calendar etc.
- **type** is type of the application that generated the event. The rest of the table is not shown in this figure.

Another table labelled “application” contains information about the different statuses a phone application can be in. While using the phone, users actually act on the different applications for example the contact list, the caller, the calendar, mailing etc. For each of these actions a record is made to keep track.
- **fk** a unique id that points to the record table
Table 3: Partial overview of the “application” table

<table>
<thead>
<tr>
<th>fk</th>
<th>event</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>34505318</td>
<td>Application.Started</td>
<td>Calendar</td>
</tr>
<tr>
<td>34505400</td>
<td>Application.Foreground</td>
<td>screensaver</td>
</tr>
</tbody>
</table>

- **event** a description of the recorded event. The field can have one of the values “Application.X” where X in {“Started”, “Foreground”, “View”, “Closed”}
- **name** The application’s name.

The Calendar is one of the phone’s applications that we want to monitor. In this purpose a table holding the same name was created to hold all the information related to the calendar. An overview of this table is showed below:

Table 4: Partial overview of the “calendar” table

<table>
<thead>
<tr>
<th>id</th>
<th>userid</th>
<th>status</th>
<th>begin</th>
<th>title</th>
<th>last_mod</th>
<th>location</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>CONFIRMED</td>
<td>20090701T090000Z</td>
<td>appoint. example</td>
<td>20090701T084546Z</td>
<td>location1</td>
<td>APPOINTMENT</td>
</tr>
<tr>
<td>2</td>
<td>202</td>
<td>TENTATIVE</td>
<td>20100801T123000Z</td>
<td>Meet Mr. dupont</td>
<td>20091023T133000Z</td>
<td>EVENT</td>
<td></td>
</tr>
</tbody>
</table>

- **id** is a unique identifier for an entry in the table.
- **userid** is a unique identifier of the users in the campaign.
- **status** is a field that can have two different value “CONFIRMED” or “TENTATIVE”
- **begin** is the scheduled time of the entry, the time is in the iCal[RFC 2445] format and the time zone was GMT+2
- **title** is the text that users insert as a denomination of the entry
- **last_mod** is the creation/modification time of the entry, the time is in the iCal format, GMT+2. Please note that changing an appointment’s location for instance will result in a new entry in the calendar table having a new last_mod value
- **location** is the text that users insert to specify a location for this entry. This field has been left empty in most of the cases.
- **type** is a field that can have one of two values “APPOINTMENT” or “EVENT”, this last value is to be used for reminders such as for birthdays and such. The “APPOINTMENT” value is meant for other kind of entries like for meetings. The rest of the table is not shown in this figure.

Another table called “calendar_history” holds basically the same type of information as shown in the Table 4. This table’s role is to keep track of all changes that happened in the calendar table.

Please note that entries can be inserted either by manually adding an event in the phone’s calendar, or by editing an existing event, or by a synchronization with the PC that resulted in importing some new events.

### 3.1.2 Some Definitions

Before presenting the results of the data-mining, let’s first define some important notions.

- **ACTIVE calendar Users**: We define active users as users who insert manually APPOINTMENTS in their phones. In contrast PASSIVE calendar users are those who use their calendar only for checking their appointments, probably their calendar have been populated by synchronization.

- **ENTRY**: a database entry, this word makes a reference to a tuple in one of the tables above. An entry in the “calendar” table can be obtained by synchronisation or by manual insert in the phone.

  - **EVENT**: one of the types of the entries in the calendar table. EVENT means that it is most likely a reminder/anniversary etc.
  
  - **APPOINTMENT**: another type of the entries in the calendar table. APPOINTMENTS is the category that includes meetings.
MEETING: A meeting is a theoretical notion. We define a meeting in the context of this database mining as having two or more users meet either physically or via phone conferences. We decided for this project, and considering the database potential, to define a meeting as one of the cases below:

- Two or more users have an entry in their calendar with the same begin time and the same title.
- Two or more users have an entry in their calendar with the same begin time and their bluetooth devices detect each others.

3.1.3 Performed work and encountered limitations

Mining this database was far from being a straightforward task. We encountered a lot of limitations, anomalies, and strange behaviours, for which we had to come up with solutions, workarounds, and explanations.

Limitations The GUI of the phone’s calendar application, as shown in figure 5, allows users to choose the duration of their appointments. However the duration field was not present in the database, somehow the data collection software just ignored it. This information would have allowed us to build probability distribution of meeting length, and thus make our model look even more realistic. We tried inferring length of appointments using additional resources such as bluetooth, however implementing such workarounds need a lot of time and has weak accuracy (High amount of false positives)

The structure of some parts of the database was incoherent. The bluetooth information for instance was spread on 5 tables with a lot of redundancy, making the task of building efficient, filtered requests somewhat tedious.

The calendar table contained a lot of duplicates because each time the user updated an appointment or an event (e.g. by changing the title) this resulted in the creation of a new entry that had almost the same data except the minor change that user performed. We filtered out the duplicates by taking only the entry with newest “last_mod” value.

The type of fields was sometimes not appropriate. For instance, in the database we find three representations of time: TIMESTAMP, LINUX time and iCal format. We encountered big problems while trying to perform requests concerning the comparison of times in different tables, basically the iCal format was encoded as text while the other two are “date”, we had to convert on the fly from one type to another, taking in account the fact that the server time (i.e. the time in record table) and the phone time had two different encodings and two different time zones.

Unexpected results Once the mining started showing results we observed a peculiar peak at 7:00 am in the diagram of distribution of appointments over day hour. Later on we discovered that the phone’s default time for an appointment or event is 08:00 am and since the server was one hour less in time zone than the phones this led to the peak. We chose to ignore all entries at 7:00 am because of this bias, moreover we focused on the period between 09:00 am to 10:00 pm to plot the charts. Entries that had begin time not in this interval were redistributed inside the interval to increase the probability distribution.

Workarounds To identify a meeting in the database we looked for the presence of one of the two cases listed in Section 3.1.2.

The first method was quite easy to implement, although the results were biased because a majority of the meetings belonged to two users. The second method did not succeed, because the bluetooth information was spread in 5 tables with a lot of redundancy and a request for a unique users was taking enormous time to execute. A workaround would have been to implement a view that contains the temporary query results and than to perform further queries on it. Using bluetooth can help in studying user preferences and meeting patterns, however one needs first to consider the user habits and social affinities [14], for instance to distinguish frequently seen bluetooth devices as places/persons that user go to /meet.

To be able to get the portion of active users we used the tables “application”, “records”. The solution we implemented used the fact that if a user creates an entry from his phone’s calendar, the time of creation of the event and time the calendar application was being opened (See Table 3) is almost simultaneous.
3.2 Results

We will present in the following some of the results that were most interesting, the results are separated into two main areas: Calendar usage and Appointment Scheduling.

3.2.1 Phone calendar usage

This section shows mostly results about users habits regarding their phones, and more specifically how they use their calendar application on mobile phones. The first approach was to see how many people use calendar application in general, meaning either actively creating entries on the phone or by synchronizing with PC and only viewing on the phone.

<table>
<thead>
<tr>
<th>Total users</th>
<th>187</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calendar users</td>
<td>57</td>
</tr>
<tr>
<td>Rate</td>
<td>30.48%</td>
</tr>
</tbody>
</table>

Figure 6: Users having used their calendar application at least once

On the portion of people that are using the calendar (i.e. 31%, Figure 6), it was interesting to investigate how many times people do actually interact with their calendar weekly.

The figure 7 shows that in average more than 50% of the users open or interact with their phone calendars at the maximum once a day.

Now let’s look at the set of users that actively interact with their calendar, meaning that they manually introduce events or appointments in their phones.

<table>
<thead>
<tr>
<th>Total users</th>
<th>187</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calendar active users</td>
<td>10</td>
</tr>
<tr>
<td>Rate</td>
<td>5.35%</td>
</tr>
</tbody>
</table>

Figure 7: Weekly interaction rate with calendar

Figure 8: Proportion of calendar active users

The figure 8 shows that on the total of 187 users, only around 6% were actively inserting entries in the phone. Actively means that this 6% of users are actually inserting manually on the phone their appointments.
3.2.2 Appointment scheduling

This part of the data mining focused on understanding the scheduling patterns.

At the time this report was made the database contained 187 users, the results shown below are based on this user population. The average of entries in the calendar table was 66.42 entry/p.user this number means that in average an user introduced or modified an entry his calendar up to 66 times during the campaign duration, and the users subscribing period was about 24.36 weeks in average. As a result an user modifies his calendar 2.75 times per week in average.

![Figure 9: Clusters of users according to number of entries](image)

The figure 9 shows the clusters of users based on the number of entries they had on their respective calendars. Please note that the number of users having entries in their phone is greater than in Figure 6, this is due to synchronization software that may import calendar entries to the phone, even if the user doesn't view them.

The graphs in Figure 10 show the distribution of appointments over weekdays and day hours. As expected the week ends have lower scheduling, and the values for the other days is almost equally distributed. A more interesting figure is the distribution of appointments over day hours (Figure 10, right), it was a particularly biased diagram in the beginning due to the fact that participants were using their calendars as reminders, this was dealt with by filtering out the entries by their type, focusing on "APPOINTMENT".

![Figure 10: Distribution of appointments per week days and day hours](image)

Going further we built the probability distribution of appointments per day hour. We will come back to this table in a later chapter to explain the use of it.

<table>
<thead>
<tr>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.07%</td>
<td>9.18%</td>
<td>7.19%</td>
<td>8.16%</td>
<td>8.18%</td>
<td>10.30%</td>
<td>7.16%</td>
<td>7.44%</td>
<td>7.96%</td>
<td>8.82%</td>
<td>7.16%</td>
<td>4.61%</td>
<td>1.50%</td>
<td>2.32%</td>
</tr>
</tbody>
</table>

Table 5: Probability distribution over day hours
3.3 Discussions

The results from the section 3.2.2 show that, as one would expect, appointments scheduling is shaped by the time of day. Indeed the graph in Figure 10 show that entries in users calendars tend to be less numerous in the early morning or late evening.

For the part concerning phone calendar usage in section 3.2.1, the results were more surprising. The fact is that only around 6% of all the users were actually using the phone calendar to manually insert appointments. This can be explained by the fact that N95 phones have reduced keypad making it hard and not very intuitive to use nor to insert text. Another explanation is that appointment scheduling in an enterprise is mostly made via centralized servers meaning that users often use their PCs at work or at home to schedule and their phone to view only.

Now that we have extracted the distribution of appointments scheduling, we can use it to shape generated problems and give them a real life dimension.

4 The problem generator

This section describes the problem generator that was implemented in order to generate real-life looking problems, which will be the input of the algorithms described in Section 2.2.

4.1 Arbitrary, Satisfaction Problems

A first version of the problem generator creates arbitrary problems with the structure of scheduling problems. We give in the following a description of the these problems according to the PEAV mapping scheme (See Section 2.2):

- The set of variables $X$ is: $X = \{ X_{a,1}, X_{b,1}, ..., X_{z,i} \}$ where $X_{m,n}$ is the starting time of the meeting number $m$ of the agent $n$
- The set of agents $A = \{ a_1, ..., a_k \}$
- The set of domains $D = \{ d_1 \}$ where $d_1 = \{ 9, ..., 22 \}$
- The set of constraints $C$, here we have:
  - binary equality constraints between all the variables of agents that participate in a certain meeting.
  - binary non-equality constraints between all the variables of a single agent, expressing that no two meetings have the same starting time.

In a first phase scheduling problems where generated in a random manner, by providing:

- The number of meetings to schedule
- The number of participants per meeting
- The total number of participants in the organization

The problem generator was able to output an instance in the form of an XCSP file. The XCSP could then be passed to the solver part (FRODO [15]), which would return a feasible assignment to the variables.

4.2 Real-life-looking Satisfaction Problems

As mentioned in the data-mining section, the probabilities that were extracted from the database can serve as a base to improve the problems output by the problem generator.

The real-life problems generator generates problem instances that have some more properties e.g. pre-filled-slots. Unfortunately we could not include other properties such as meeting duration and number of participants per meeting, because on the one hand the information about meeting durations was absent from the database, and on the other hand the number of participants in the campaign was so low that it was quite impossible to infer any details about the number of participants in a meeting.

To the formal description of our meeting scheduling problems we add the following set of unary constraints:

- unary constraints reflecting the preexisting meetings. For each agent, after randomly selecting some slots $S_r$, we generated the new unary constraints as follows: for each of the variables of an agent, the new constraint added an infinite cost to the slots in $S_r$

The rate of filling of schedules is a decimal number in $[0, 1]$, representing the initial pre-filling of the agents’ schedules. For instance if a rate of 0.3 is given, then the participants schedules will be pre-filled at 30%. The way schedules are pre-filled is the following:
- let $P$ be the number of slots that will be pre-filled
- For each agent, choose randomly $P$ slots from all the slots and fill it with a meeting

Choosing randomly means that slots are drawn without replacement from the probability distribution in Table 5.

4.3 Real-life-looking Optimisation Problems

The latest step of the implementation was to be able to generate distributed optimization problems. Meeting scheduling problems can be viewed as optimization problems if one consider the agents’ preferences. The real-life looking problems are scheduling problems augmented with preferences. We injected preferences in our problems by looking at Table 5 and considering that the probability of having a meeting at a certain slot $t$ reflects the value of this slot to the agent: the higher the probability, the more valuable the slot and the higher its cost.

The bottom line is that for every slot $s \in [9, 22]$, we assign a cost $c(s) = 1 - P(s)$, where $P(s)$ is the probability that $s$ is occupied.

These costs were included in the problem by modifying the unary constraint of each agent and adding new tuples that give the cost of each slot that is non occupied.

5 Experimental results

This section presents some experimental results that highlight the performance of the algorithms mentioned (Section 2.2). All experiments were run on a machine with Intel Xeon Core with 2.83 GHZ and 3.25GB of RAM, the JVM environment was launched with 1.6GB heap space, we used Eclipse IDE Galileo.

The experiments were run as follows:

foreach of the parameters [agents, meetings, agents per meeting, rate]
create a 100 instances
run every instance in all algorithms and save output.

After running the experiments, the 100 results for each parameter were stored in an ordered list. Then the median was computed, with 95% confidence intervals over every 100 experiments.

5.1 Effect of the number of agents

Remarks:
- **Data size**: Increasing the number of agents leads to a decrease of the number of meetings per agent. Which in turn leads to a decrease in the induced width, making the UTIL messages of smaller size (Figure 12(a)).
- **Messages exchanged**: ADOPT performs worst in this case (Figure 12(b)); this is due to the combinatorial explosion of messages as in ADOPT each node sends more messages than it receives.
- **Time Needed**: The difference of time needed is noticeable: P-DPOP’s time is greater by 2 orders of magnitude compared that those of ADOPT and SynchBB (Figure 12(c)). Please note however that the problems in the figures do not include preexisting meetings, which means that the constraints are very loose and easy to satisfy, which advantages ADOPT and SynchBB (cf. Section 5.4).
5.2 Effect of the number of meetings

Remarks:

- **Data size**: When increasing the number of meetings the problems get more complicated (more constraints per agent). Which logically leads to an increase in the size of messages for (P)DPOP, while ADOPT’s and SynchBB’s messages sizes remain constant (Figure 13(a)).

- **Messages exchanged**: ADOPT sends smaller messages in size, but their number grows exponentially (Figure 13(b)) when the problem becomes harder to solve. DPOP is the one that performed the best in this case.

- **Time needed**: This is a surprising result as the difference between P-DPOP and SynchBB is about 4 orders of magnitude (Figure 13(c)), however it can easily be explained by the fact that once SynchBB (as well as ADOPT) finds a solution the algorithm stops, whereas (P)DPOP continues until all the possibilities are evaluated.

5.3 Effect of the number of agents per meeting

Remarks

- **Data size**: The number of agents per meeting drastically influences the complexity of the problems. The data exchanged for all algorithms grows with the number of agents, however for ADOPT and P-DPOP the change is exponential (Figure 14(a)). Only SynchBB was able to solve problems with a number of agents per meeting equal to 4 (P-DPOP ran out of memory).

- **Messages exchanged**: The slope of messages exchanged graphs is bigger for ADOPT than for other algorithms. (Figure 14(b))

When further increasing the number of agents per meeting, the results are harder to get for DPOP and P-DPOP. A lot of out of memory runs made it impossible to get the minimum of 71 results to compute the median.

5.4 Effect of preexisting meetings

Recall that we represent the variation of existing meeting as a rate of pre-filling the schedules. For instance a rate of 0.3 means that 30% of the agents’ schedules were prefilled with meetings.
5.5 Effect of adding preferences

In this experiment we varied the number of meetings as in section 5.2, and in addition we set the rate of pre-filling schedules to be 30%. Moreover we set for each agent a cost for each of the slots in the domain. The cost of slot \( s \) for agent \( A \) being

\[
c(s) = 1 - P
\]

where \( P \) is the probability of having a meeting at the time slot \( s \) according to the Table 5.

Remarks

- **Data size**: We see an increase in the slope of ADOPT in Figure 16(a) compared to Figure 13(a), and this is due to the fact that finding the optimal solution implies more runs. In the other hand DPOP has almost no change in data size compared to Figure 13(a) as it perceives no difference in the problems with and without preferences due to dynamic programing.

- **Number of messages**: An outcome of the addition of preferences to the problems is that ADOPT and SynchBB has to compute more solutions than in the satisfaction problems, this is why we see an exponential growth of the number of messages sent (Figure 16(b)). Whereas (P)DPOP has absolutely no change compared to Figure 13(b).

- **Simulated time**: In the Figure 16(c) we see that SynchBB and ADOPT perform not as good as in the Figure 13(c), due the addition of preferences and to the preexisting meetings, the simulated time for ADOPT had increased by one order of magnitude. And it is probably going to be even more if the rate of pre-filling the schedules is around 50% (See figures 14(a),14(b)).
6 Conclusion and future work

We have presented a random problem generator that produces meeting scheduling instances augmented with preferences and preexisting meetings. Unlike other approaches [9, 12, 11], our model uses a preference estimation that is globally inferred from a database of existing meetings. The generated problems served in a following step to experiment P-DPOP, that hadn’t been tested so far in real-life situations, and to compare its performance with a set of other DCOP algorithms. The experimental results show that P-DPOP performs poorly regarding the message size (exponential growth), which is the cost to pay to protect the privacy of the agents. In the other hand ADOPT showed to be severly affected by the addition of user preferences and preexisting meetings to the generated problems.

Future work may include improving the problem generator with variable time slot length and preferences based on negotiation behaviors. Regarding P-DPOP we can think about reducing the redundancy of information in the UTIL messages by having n-ary equality constraints and exploiting the symmetries in the UTIL messages. This would reduce drastically the data size, making this algorithm fit better the needs of today’s meeting scheduling with privacy guarantees. Another technical improvement would be to include a new heuristic method in FRODO that basically groups all the variables of the same agent in such a way that most messages are virtual, intra-agent messages; this way the number of actual messages sent would be reduced.

Figure 14: Results for the variation of the number of agents per meeting, fixing: meetings= 3, agents=10, rate=0
Figure 15: Results for variation of preexisting meetings rate, fixing: meeting= 3, apm=2, agents=5

References


Figure 16: Results for variation of number of meetings with preferences, fixing: apm=3, agents=5, rate=0.3


