Scheduling B2B meetings

Miquel Bofill*, Joan Espasa*†, Marc Garcia, Miquel Palahí*‡, Josep Suy*, and Mateu Villaret*

Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain
{mbofill,jespasa,mgarciao,mpalahi,suy,villaret}@imae.udg.edu

Abstract. In this work we deal with the problem of scheduling meetings between research groups, companies and investors in a scientific and technological forum. We provide a CP formulation and a Pseudo-Boolean formulation of the problem, and empirically test the performance of different solving techniques, such as CP, lazy clause generation, SMT, and ILP, on industrial and crafted instances of the problem. The solutions obtained clearly improve expert handmade solutions with respect to the number of idle time slots and other quality parameters.

1 Introduction

Business-to-business (B2B) events typically provide bilateral meeting sessions between participants with affine interests. These B2B events occur in several fields like sports, social life, research, etc. In this paper we describe the application developed to generate the timetable of such meetings in the *4th Forum of the Scientific and Technological Park of the University of Girona*.[1] The goal of this forum is to be a technological marketplace in Girona by bringing the opportunity to companies, research groups, investors, etc., to find future business partnerships.

The scheduling of the meetings is a tough task and requires expertise at distinct levels: on the one side, the human matchmaker should choose the appropriate matches maximizing somehow the potential results of the meetings according to the interests of the participants. On the other hand side, the timetable generation must satisfy several constraints, e.g., avoid meeting collisions, avoid unnecessary idle time between meetings for each participant or too many meeting location changes, etc.

In previous editions of the forum, and in other events with B2B meetings held in the park of the University of Girona, the human matchmaker made both tasks by hand. This process showed to be highly unsatisfactory with respect to

*Supported by the Spanish Ministry of Science and Innovation (project TIN2012-33042).

†Supported by UdG grant (BR 2013).

‡Supported by UdG grant (BR 2010).

[1] http://www.forumparcudg.com

the human effort required and also with respect to the final result obtained. Moreover, the growing participation makes the work much harder.

As far as we know, there are no many works dealing with this problem. On the one hand, in [12] we can find a system that is used by the company *piranha womex AG* for computing matchmaking schedules in several fairs. This system differs from ours in some aspects. For instance, it does not consider forbidden time slots but unpreferred ones, and it allows meeting collisions under the assumption that the companies can send another participant. Moreover, it is based on answer set programming, whereas we follow a different model-and-solve approach. On the other hand, the system B2Match [1] is a commercial system which does not support location change minimization.

Our work focuses on the generation of the meetings' timetable. That is, the human matchmaker provide us with the meetings that she wants to be scheduled and we generate the timetable. We provide both a CP model and a Pseudo-Boolean model for the problem, and compare the efficiency of different solving techniques such as CP, lazy clause generation, SMT, and ILP, on several real instances.

The obtained results for this year's edition of the Forum of the Scientific and Technological Park of the University of Girona have been very satisfactory. Moreover, using instances of previous years we have observed that the number of idle time slots could be dramatically improved.

The rest of the paper is structured as follows. In Section 2 we define the problem at hand. In Section 3 we present the different models considered, and in Section 4 we check the efficiency of distinct solvers on these models, using real instances. Conclusions are given in Section 5.

2 The B2B problem

In the Forum of the Scientific and Technological Park of the University of Girona, the participants answer some questions about their interests and expertise, in the event registration phase. This information is made public to the participants, who may ask for bilateral meetings with other participants with a (normal or high) priority. They also indicate their time availability for the meetings (notice that in the forum there is also a conference with talks given by the participants and therefore they should reserve some time slots for their own talks). Moreover, the participants may ask for meetings in the morning or the afternoon session. Then, according to this information (participants availability and priorities for the desired meetings), the human matchmaker proposes a set of matches (meetings) to be scheduled. Once this schedule is obtained, the matchmaker asks the participants for confirmation of the meetings timetable. Then, the confirmed ones are fixed and the rejected ones are retracted. With this partial timetable, and considering the late arrivals of participants interested in having some meeting, the expert tries to add other meetings in the timetable by individually contacting with the participants. Finally, the distribution of tables is made. Below we formally define the basic problem.

Definition 1. *Let P be a set of participants, T a list of time slots and L a set of available locations (tables). Let M be a set of unordered pairs of participants in P (meetings to be scheduled). Additionally, for each participant $p \in P$, let $f(p) \subset T$ be a set of forbidden time slots.*

A feasible B2B schedule *S is a total mapping from M to $T \times L$ such that the following constraints are satisfied:*

- *Each participant has at most one meeting scheduled in each time slot.*

$$\forall m_1, m_2 \in M \text{ such that } m_1 \neq m_2 : \\ \pi_1(S(m_1)) = \pi_1(S(m_2)) \implies m_1 \cap m_2 = \emptyset \quad (1)$$

- *No meeting of a participant is scheduled in one of her forbidden time slots.*

$$\forall p \in P, m \in M : \\ p \in m \implies \pi_1(S(m)) \notin f(p) \quad (2)$$

- *At most one meeting is scheduled in a given time slot and location.*

$$\forall m_1, m_2 \in M \text{ such that } m_1 \neq m_2 : \\ \pi_1(S(m_1)) = \pi_1(S(m_2)) \implies \pi_2(S(m_1)) \neq \pi_2(S(m_2)) \quad (3)$$

The B2B scheduling problem *(B2BSP) is the problem of finding a feasible B2B schedule.*

The B2BSP is clearly in NP, and can be easily proved to be NP-complete by reduction from the restricted timetable problem (RTT) in [11].

Typically, we are interested in schedules that minimize the number of idle time periods. By an *idle time period* we refer to a group of idle time slots between a meeting of a participant and her next meeting. Before formally defining this optimization version of the B2BSP, we need to introduce some auxiliary definitions.

Definition 2. *Given a B2B schedule S for a set of meetings M, and a participant $p \in P$, we define $L_S(p)$ as the list of meetings in M involving p, ordered by its scheduled time according to S:*

$$L_S(p) = [m_1, \ldots, m_k], \text{ with} \\ \forall i \in 1..k : p \in m_i \\ \forall m \in M : p \in m \Rightarrow \exists! i \in 1..k : m_i = m \\ \forall i \in 1..(k-1) : \pi_1(S(m_i)) < \pi_1(S(m_{i+1}))$$

By $L_S(p)[i]$ we refer to the i-th element of $L_S(p)$, i.e., m_i.

Definition 3. *We define the* B2B Scheduling Optimization Problem (B2BSOP) *as the problem of finding a feasible B2B schedule S, where the total number of idle time periods of the participants is minimal, i.e., minimizes*

$$\sum_{p \in P} \#\{L_S(p)[i] \mid i \in 1..|L_S(p)| - 1, \pi_1(S(m_i)) + 1 \neq \pi_1(S(m_{i+1}))\} \quad (4)$$

It is also quite common to ask for the minimization of location changes between consecutive meetings of the same participants. Due to the requirements of the event organization (there is a phase where the human matchmaker manually arranges new meetings by reusing time slots and participants' availabilities), we do this minimization with the time slots of the meetings already fixed.

Definition 4. *We define the* B2B location scheduling optimization problem (B2BLOP) *as the problem of, given a B2B schedule S, re-assign to each meeting a location in a way such that the total number of location changes for consecutive meetings of the same participant is minimal, i.e., minimizes*

$$\sum_{p \in P} \#\{L_S(p)[i] \mid$$
$$i \in 1..|L_S(p)| - 1, \pi_1(S(m_i)) + 1 = \pi_1(S(m_{i+1})), \pi_2(S(m_i)) \neq \pi_2(S(m_{i+1}))\}$$

As an additional constraint, we consider the case where meetings may have a morning/afternoon requirement, i.e., that some meetings must necessarily be celebrated in the morning or in the afternoon. Let's then consider that the set of time slots T is divided into two disjoint sets T_1 and T_2 and, moreover, that we have a mapping t from meetings m in M to $\{1, 2, 3\}$, where 1 means that the meeting m must take place at some time slot in T_1, 2 means that it must take place at some time slot in T_2, and 3 means that it does not matter. Then the schedule should also satisfy the following requirement:

$$\forall m \in M : \\ (t(m) = 1 \implies \pi_1(S(m)) \in T_1) \ \wedge \ (t(m) = 2 \implies \pi_1(S(m)) \in T_2) \quad (5)$$

Summing up, the scheduling process goes as follows:

1. The human matchmaker arranges the meetings taking into account the participants' requirements and preferences (who they want to meet, with which priority, and a possible restriction to the morning or afternoon frame for the meeting) and their forbidden hours (for example, the hours where the participant is giving a talk at the event).
2. We solve the B2BSOP with the chosen meetings.
3. The human matchmaker removes the revoked meetings (for instance, one participant may not be interested in a meeting requested by another participant) from the solution found in Step 2, and manually adds new last-arrival meetings.
4. We solve the B2BLOP with the meetings proposed in Step 3.

3 Models

It is our aim to model the aforementioned problem by means of a declarative CP language and evaluate the performance of several solvers taking this formulation as input. To this purpose, MiniZinc [15] is almost a perfect target, as it is a

medium-level solver-independent CP modelling language, which is supported by an increasing number of backend solvers, constituting a de facto standard.

For the sake of completeness, we also consider WSimply [6], a system with a similar language to that of MiniZinc, but supporting weighted constraints and several predefined meta-constraints which allow to express, e.g., homogeneity in violations, in order to enforce fairness of solutions. A proposal for incorporating similar ideas into MiniZinc has been presented in [5].

Moreover, since a priori it is not clear if a high-level or a low-level model will be better in terms of performance, we have considered two different models: a CP model (with finite domain variables like t_i, denoting the time at which meeting i takes place) and a Pseudo-Boolean model (with 0/1 variables like $x_{i,j}$, stating that meeting i is celebrated at time j).

3.1 A WCSP model for the B2BSOP

Here we give the WSimply version of the CP model for the B2BSOP, expressed as a weighted CSP.

Parameters

```
int nParticipants;                      % # of participants
int nMeetings;                          % # of meetings
int nTables;                            % # of locations
int nTimeSlots;                         % # of time slots
int nMorningSlots;                      % # of morning time slots
int meetings[nMeetings,3];              % Meetings to schedule
int tnForbidden;                        % Total # of forbidden slots
int forbidden[tnForbidden];             % Forbidden slots
int indexForbidden[nParticipants+1];    % Start indices in forbidden
int nMeetingsParticipant[nParticipants]; % # of meetings of each part.
```

The number of afternoon time slots is `nTimeSlots` − `nMorningSlots`, and for this reason it is not explicitly defined.

The `meetings` matrix has three columns, denoting the number of the first participant, the number of the second participant and the type of session required (1: morning, 2: afternoon, 3: don't care) for each meeting to be scheduled.

The `indexForbidden` array contains the indices where the forbidden time slots of each participant do begin in the `forbidden` array. This array has an extra position in order to ease the modelling of the constraints (see the `Forall` statements below), with `indexForbidden[nParticipants+1] = tnForbidden+1`.

Variables and domains

```
Dom dTimeSlots = [1..nTimeSlots];
Dom dUsedSlots = [0..1];
Dom dTables    = [0..nTables];
```

```
IntVar schedule[nMeetings]::dTimeSlots;
IntVar tablesSlot[nTimeSlots]::dTables;
IntVar usedSlots[nParticipants,nTimeSlots]::dUsedSlots;
IntVar fromSlots[nParticipants,nTimeSlots]::dUsedSlots;
```

The array variable `schedule` shall contain the time slot assigned to each meeting.

The array variable `tablesSlot` will indicate the number of meetings to be celebrated at each time slot (and hence the number of tables needed). Note that by setting the domain to `[0..nTables]` we are already restricting the number of tables available.

The variable `usedSlots` is a two dimensional 0/1 array representing, for each participant i and time slot j, if i has a meeting scheduled at time j.

Finally, the variable `fromSlots` is a two dimensional 0/1 array such that, for each participant i, `fromSlots`$[i, j] = 1$ for all j from the first time slot at which a meeting for i is scheduled on.

These two last variables are used for optimization, as shown below.

Constraints

– *Each participant has at most one meeting scheduled at each time slot*, i.e., constraint (1). We force two meetings sharing one member to be scheduled at a different time slot:

```
Forall (n in [1..nMeetings]) {
  Forall (m in [n+1..nMeetings]) {
    If (meetings[n,1] = meetings[m,1] Or
        meetings[n,2] = meetings[m,1] Or
        meetings[n,1] = meetings[m,2] Or
        meetings[n,2] = meetings[m,2])
    Then { schedule[n] <> schedule[m]; };
  };
};
```

– *No meeting of a participant is scheduled in one of her forbidden time slots*, i.e., constraint (2). We force, for each meeting, to be scheduled in a time slot distinct from all forbidden time slots for both members of the meeting:

```
  Forall(j in [1..nMeetings]) {
    Forall(k in [ indexForbidden[meetings[j,1]]..
                  indexForbidden[meetings[j,1]+1]-1 ]) {
      schedule[j] <> forbidden[k];
    };
    Forall(k in [ indexForbidden[meetings[j,2]]..
                  indexForbidden[meetings[j,2]+1]-1 ]) {
      schedule[j] <> forbidden[k];
    };
  };
```

– *At most one meeting is scheduled in a given time slot and location*, i.e., constraint (3). This constraint is ensured by matching the number of meetings scheduled in time slot i with `tablesSlot`$[i]$, whose value is bounded by the number of tables available:

```
Forall(i in [1..nTimeSlots]) {
 Sum([ If_Then_Else(schedule[n] = i)(1)(0) | n in [1..nMeetings] ],
     tablesSlot[i]);
};
```

Note that we are not assigning a particular table to each meeting, but just forcing that there are enough tables available for the meetings taking place at the same time.

– *Each meeting must be scheduled in a required (if any) set of time slots*, i.e., constraint (5). Since we know the number of morning time slots, we can easily enforce this constraint:

```
Forall (n in [1..nMeetings]) {
  If (meetings[n,3] = 1) Then {schedule[n] =< nMorningSlots;}
  Else {If (meetings[n,3] = 2) Then {schedule[n] > nMorningSlots;};
  };
};
```

– *Channeling constraints.* In order to be able to minimize the number of idle time periods, i.e., objective function (4), we introduce channeling constraints mapping the variable `schedule` to the variable `usedSlots`. That is, if meeting k is scheduled at time slot j, we need to state that time j is used by both members of meeting k, by setting accordingly the corresponding value in `usedSlots`:

```
Forall(j in [1..nTimeSlots]) {
  Forall(k in [1..nMeetings]) {
    (schedule[k] = j) Implies
       (usedSlots[meetings[k,1],j] = 1 And
        usedSlots[meetings[k,2],j] = 1);
  };
};
```

In the reverse direction, for each participant e, her number of meetings as derived from `usedSlots` must match her known total number of meetings `nMeetingsParticipant`$[e]$:

```
Forall(e in [1..nParticipants]) {
  Sum([ usedSlots[e,f] | f in [1..nTimeSlots] ],
      nMeetingsParticipant[e]);
};
```

Next, we impose the required constraints on the variable `fromSlots`:

```
Forall(e in [1..nParticipants]) {
  Forall(f in [1..nTimeSlots]) {
    usedSlots[e,f] = 1 Implies fromSlots[e,f] = 1;
  };

  Forall(f in [1..nTimeSlots-1]) {
    fromSlots[e,f] = 1 Implies fromSlots[e,f+1] = 1;
  };
};
```

It is worth noting that, for any participant e, having `fromSlots`$[e, f] = 1$ for all f is possible, even if e has no meeting at time slot 1. However, the soft constraints used for optimization will prevent this from happening, as commented below.

Optimization Minimization of the objective function (4) is achieved by means of soft constraints, where a group of contiguous idle time slots between two meetings of the same participant is given a cost of 1.

Soft constraints are labeled with `Holes`$[e, f]$, where e denotes a participant and f denotes a time slot, and state that if e does not have any meeting in time slot f, but it has some meeting before, then she does not have any meeting in the following time slot:

```
Forall(e in [1..nParticipants], f in [1..nTimeSlots-1]) {
  #Holes[e,f]:((usedSlots[e,f] = 0 And fromSlots[e,f] = 1) Implies
              usedSlots[e,f+1] = 0)@{1};
};
```

We claim that, with these constraints, an optimal solution will be one having the least number of groups of contiguous idle time slots between meetings of the same participant. Note that, for each participant, we increase the cost by 1 for each meeting following some idle period. Moreover, it is not difficult to see that the (possibly null) period of time preceding the first meeting of a participant e will have no cost, since `fromSlots`$[e, f]$ can freely take value 0 for all f prior to the time of the first meeting.

Finally, since we are not only interested in optimal solutions, but in fair ones, we can add the following meta-constraint, stating that the difference between the number of idle periods of any two distinct participants is, e.g., at most 2:

```
homogeneousAbsoluteNumber([ [ Holes[e,f] | f in [1..nTimeSlots-1] ] |
                            e in [1..nParticipants] ], 2);
```

Note that this meta-constraint makes use of the labels introduced in the soft constraints. It has as first argument a list of lists ll of soft constraint labels, and as second argument a natural number n. It ensures that, for each pair of lists in ll, the difference between the number of violated constraints in the two lists is at most n. The meta-constraints supported by WSimply are described in [6].

The precise syntax and semantics of the language can be found in a technical report.[2]

The WSimply model presented above has been translated to MiniZinc in order to test the performance of a greater number of different solvers (see Section 4). The translation of hard constraints is straightforward, due to the similarities between the two languages. Soft constraints have been translated into a linear objective function, as they are not directly supported in MiniZinc. The meta-constraint used in the WSimply model has been translated into the MiniZinc constraints that would result from the translation process implemented in the WSimply compiler.

3.2 A PB model for the B2BSOP

Here we give the WSimply version of the Pseudo-Boolean model for the B2BSOP. The parameters are the same as in the CP model but, in this case, we only use 0/1 variables.

Variables and domains

```
Dom pseudo = [0..1];

IntVar schedule[nMeetings,nTimeSlots]::pseudo;
IntVar usedSlots[nParticipants,nTimeSlots]::pseudo;
IntVar fromSlots[nParticipants,nTimeSlots]::pseudo;
IntVar holes[nParticipants,nTimeSlots-1]::pseudo;
IntVar nHoles[nParticipants,5]::pseudo;
IntVar max[5]::pseudo;
IntVar min[5]::pseudo;
```

Here, the array variable `schedule` shall contain a 1 at position i, j if and only if meeting i is celebrated at time slot j.

The variables `usedSlots` and `fromSlots` are the same as in the CP case.

The variable `holes` will indicate, for each participant, when a time slot is the last of an idle period of time.

The variable `nHoles` will hold the 5-bit binary representation of the number of idle time periods of each participant, i.e., the number of groups of contiguous idle time slots between meetings of each participant.

The variables `max` and `min` will hold the 5-bit binary representation of an upper bound and a lower bound of the maximum and minimum values in `nHoles`, respectively. As we will see, these variables will be used to enforce fairness of solutions, by restricting their difference to be less than a certain value.

All variables but `schedule` are only necessary for optimization.

[2] `http://imae.udg.edu/recerca/lap/simply/docs/technical-report.pdf`

Constraints

– *Each participant has at most one meeting scheduled at each time slot*:

```
Forall (f in [1..nTimeSlots]) {
  Forall (n in [1..nParticipants]) {
    AtMost([ schedule[m,f] |
               m in [1..nMeetings],
               (meetings[m,1] = n Or meetings[m,2] = n) ], 1, 1);
  };
};
```

The `atMost`(l, e, n) global constraint (where l is a list, e is an expression of the same type of the elements in l, and n is an integer arithmetic expression) forces the number of elements in l that match e to be at most n.

– *No meeting of a participant is scheduled in one of her forbidden time slots*:

```
Forall (r in [1..nMeetings]) {
  Forall (p in [indexForbidden[meetings[r,1]]..
                indexForbidden[meetings[r,1]+1]-1]) {
      schedule[r,forbidden[p]] = 0;
  };
  Forall (p in [indexForbidden[meetings[r,2]]..
                indexForbidden[meetings[r,2]+1]-1]) {
      schedule[r,forbidden[p]] = 0;
  };
};
```

– *At most one meeting is scheduled in a given time slot and location.* We ensure this by forcing that no more than `nTables` meetings are scheduled in the same time slot:

```
Forall(f in [1..nTimeSlots]) {
  AtMost([schedule[n,f] | n in [1..nMeetings]],1,nTables);
};
```

– *Each meeting must be scheduled in the required (if any) set of time slots.* By means of sums, we force that each meeting is scheduled exactly once in its required set of time slots:

```
Forall (n in [1..nMeetings]) {
  If (meetings[n,3] = 1) Then {
    Sum([ schedule[n,f] | f in [1..nMorningSlots] ], 1);
    [ schedule[n,f] = 0 | f in [nMorningSlots+1..nTimeSlots] ];
  } Else {
  If (meetings[n,3] = 2 ) Then {
    [ schedule[n,f] = 0 | f in [1..nMorningSlots] ];
    Sum([ schedule[n,f] | f in [nMorningSlots+1..nTimeSlots] ], 1);
  } Else {
    Sum([ schedule[n,f] | f in [1..nTimeSlots] ], 1);
  }};
};
```

Note that list comprehensions can be used to post constraints.

– *Channeling constraints.* The channeling constraints are analogous to before:

```
Forall(j in [1..nTimeSlots]) {
  Forall(k in [1..nMeetings]) {
    (schedule[k,j] = 1) Implies
       (usedSlots[meetings[k,1],j] = 1) And
       (usedSlots[meetings[k,2],j] = 1));
  };
};

Forall(e in [1..nParticipants]) {
  Sum([ usedSlots[e,f] | f in [1..nTimeSlots] ],
      nMeetingsParticipant[e]);
};

Forall(e in [1..nParticipants]) {
  Forall(f in [1..nTimeSlots]) {
    usedSlots[e,f] = 1 Implies fromSlots[e,f] = 1;
  };

  Forall(f in [1..nTimeSlots-1]) {
    fromSlots[e,f] = 1 Implies fromSlots[e,f+1] = 1;
  };
};
```

Optimization The soft constraints are the same as in the CP model:

```
Forall(e in [1..nParticipants], f in [1..nTimeSlots-1]) {
  ((usedSlots[e,f] = 0 And fromSlots[e,f] = 1) Implies
  usedSlots[e,f+1] = 0)@{1};
};
```

The homogeneity meta-constraint `homogeneousAbsoluteNumber` used in the CP model cannot be used in the Pseudo-Boolean model since, in its current implementation, WSimply will translate this meta-constraint into a set of non (Pseudo-)Boolean constraints. For this reason, this meta-constraint needs to be simulated here. We proceed as follows.

On the one hand, we post the constraints for the array variable `holes` which contains, for each participant, the last time slot of each idle period of time:

```
Forall(e in [1..nParticipants], f in [1..nTimeSlots-1]) {
  (usedSlots[e,f] = 0 And fromSlots[e,f] = 1 And usedSlots[e,f+1] = 1)
      Implies holes[e,f]=1;
  holes[e,f]=1 Implies
  (usedSlots[e,f] = 0 And fromSlots[e,f] = 1 And usedSlots[e,f+1] = 1);
};
```

On the other hand, we post the constraints defining the 5-bit binary representation of the number of idle time periods of each participant (the `sum1` function returns the sum of the elements in the list it receives as argument):

```
Forall(e in [1..nParticipants]){
  sum1([ holes[e,f] | f in [nTimeSlots-1] ]) =
  16*nHoles[e,1]+8*nHoles[e,2]+4*nHoles[e,3]+2*nHoles[e,4]+nHoles[e,5];
};
```

Finally, we constrain the difference between the number of idle time periods of any two distinct participants to be at most 2. We do this by constraining the difference between an upper bound and a lower bound of the maximal and minimal number, respectively, of idle time periods of all participants, to be at most 2:

```
Forall(e in [1..nParticipants]){
  16*max[1]+8*max[2]+4*max[3]+2*max[4]+max[5] >=
  16*nHoles[e,1]+8*nHoles[e,2]+4*nHoles[e,3]+2*nHoles[e,4]+nHoles[e,5];

  16*nHoles[e,1]+8*nHoles[e,2]+4*nHoles[e,3]+2*nHoles[e,4]+nHoles[e,5]
  >= 16*min[1]+8*min[2]+4*min[3]+2*min[4]+min[5];
};

2 >= 16*max[1]+8*max[2]+4*max[3]+2*max[4]+max[5] -
     16*min[1]+8*min[2]+4*min[3]+2*min[4]+min[5];
```

4 Experiments and comparisons with manually generated solutions

In this section we compare the performance of several state-of-the-art CSP, WCSP, ILP and PB solvers with the proposed models. We also analyse which is the improvement of our solution over some handmade solutions from past editions of the forum.

As said, apart from the two (CP and PB) WSimply models presented, we have also considered a MiniZinc model in order to test the performance of a greater number of different solvers. The MiniZinc model considered is an accurate translation of the first WSimply (CP) model. All models and data used in this paper can be found in `http://imae.udg.edu/recerca/lap/simply/`.

For solving the WSimply instances we have used the SMT solver Yices 1.0.33 [10] through its API, with two different solving methods:

- WPM1, as described in [6]. This is an adaptation of the algorithms introduced in [7,14] for (weighted) MaxSAT to the context of weighted MaxSMT.
- SBDD, as described in [9]. In this approach, the weighted SMT constraints are replaced by a linear objective function, which is encoded as a shared BDD using the compact and generalized arc-consistent encoding of [3].

We have also considered the translation of the CP model written in WSimply into ILP, and used IBM ILOG CPLEX 12.6 for solving the resulting instances. The translation of high-level CP models into ILP is a new feature supported by the WSimply system, using similar transformations to that applied for the PB case, as described in [8].

For solving the MiniZinc instances we have used Gecode 4.2.1 [17], a state-of-the-art CP solver, and Opturion 1.0.2 [2], winner of the 2013 MiniZinc Challenge[3] in the free search category, using lazy clause generation [16].

For solving the PB WSimply instances we have used CPLEX 12.6, SCIP 3.0.1 [4] and clasp 3.1.0 [13]. The two former obtained the best results in the last PB competition[4] in the category "optimisation, small integers, linear constraints", which fits our problem. However, the latter has shown a much better performance on this problem. The transformations used to obtain plain PB instances from WSimply are described in [8].

All experiments have been run on a cluster of Intel® Xeon™CPU@3.1GHz machines, with 8GB of RAM, under 64-bit CentOS release 6.3, kernel 2.6.32, except for the experiments with Opturion, where we have used a slightly different computer (Intel® Core™CPU@2.8GHz, with 12GB of RAM, under 64-bit Ubuntu 12.04.3, kernel 3.2.0) due to some library dependence problems. We have run each instance with a cutoff of 2 hours.

We have considered four industrial instances provided by the human expert: instances tic-2012a and tic-2013a, from past editions of a technology and health forum, and instances forum-2013a and forum-2014a, from the previous year and this year's editions of the scientific and technological forum.

Taking as basis these four instances, we have crafted five more instances of distinct hardness by increasing the number of meetings, reducing the number of locations and changing the morning/afternoon preferences of some meetings. Table 1 summarizes the results of the experiments.

Looking at the results, it can be said that clasp is the best solver on the easier instances (only beaten by CPLEX in two cases), and SBDD is the best on the harder instances, both of them using conflict driven solvers. The latter is also the most robust method when considering all instances, with a good compromise between solving time and quality of the solutions. Due to the fact that SBDD is based on representing the objective function as a BDD, and iteratively calling the decision procedure (an SMT solver) with successively tighter bounds, we can obtain a feasible solution at each stage of the optimization process. The WPM1 method also exhibits good performance on many instances, but it cannot provide suboptimal solutions as, roughly, it approaches to solutions from the unsatisfiable side.

An interesting aspect is that, in all real instances from past editions (tic-2012a, tic-2013a and forum-2013a) we obtained an optimum of 0 idle time periods between meetings of the same participant, whereas the expert handmade solutions included 20, 39 and 99 idle time periods respectively, as shown in Table 2. Note also that only for some crafted instances, and the bigger real instance from the last forum, we could not certify the optimality of the solutions found within two hours.

In Section 3 we have not included the model used for the B2BLOP due to lack of space. However, one aspect of the model that deserves a comment is the

[3] `http://www.minizinc.org/challenge2013/results2013.html`

[4] `http://www.cril.univ-artois.fr/PB12/`

Table 1. Solving time (in seconds) and optimum found (number of idle time periods) per instance, model and solver. The instances are tagged with (#meetings, #participants, #locations, #time slots, #morning time slots). TO stands for time out and MO for memory out. The cutoff is 2 hours. For aborted executions we report the (sub)optimum found if the solver reported any. Best running times and upper bounds of the objective function are indicated in boldface.

Instance	CP Model										PB Model					
	WSimply						MiniZinc				WSimply					
	WPM1		SBDD		CPLEX		Gecode		Opturion		CPLEX		SCIP		clasp	
tic-2012a (125,42,21,8,0)	2.0	0	2.7	0	3375.6	0	1.4	0	8.1	0	7.3	0	126.1	0	**0.1**	**0**
tic-2012c (125,42,16,8,0)	51.1	0	235.4	0	TO	4	TO	-	2206.2	0	**18.0**	**0**	1692.7	0	977.9	0
tic-2013a (180,47,21,10,0)	25.0	0	65.2	0	TO	9	2923.1	0	394.5	0	1345.3	0	TO	13	**2.1**	**0**
tic-2013b (184,46,21,10,0)	5.6	0	24.1	0	TO	8	3.4	0	108.0	0	121.0	0	4975.4	0	**2.1**	**0**
tic-2013c (180,47,19,10,0)	TO	-	TO	8	TO	18	TO	-	TO	8	**TO**	**4**	TO	31	TO	-
forum-2013a (154,70,14,21,13)	5542.3	0	3128.1	0	MO	83	TO	-	1142.8	0	TO	20	TO	-	**52.4**	**0**
forum-2013b (195,76,14,21,13)	TO	-	**TO**	**12**	TO	-	TO	-	TO	50	MO	-	TO	-	TO	24
forum-2013c (154,70,12,21,13)	TO	-	**TO**	**20**	MO	50	TO	-	TO	23	TO	30	TO	-	TO	-
forum-2014a (302,78,22,22,12)	TO	-	**TO**	**7**	TO	-	TO	-	TO	90	MO	-	TO	-	TO	-

use of meta-constraints, to ensure fairness in the number of location changes of the participants. With the meta-constraint

```
homogeneousAbsoluteNumber([[Changes[e,f] | f in [1..nTimeSlots-1]]
                          | e in [1..nParticipants]], Hfactor);
```

the user can look for solutions where the difference on the number of location changes between participants is at most `HFactor`, and with the meta-constraint

```
maxCost([[Changes[e,f] | f in [1..nTimeSlots]]
         | e in [1..nParticipants]], MaxChanges);
```

the user can look also for solutions where the number of location changes per participant is at most `MaxChanges`.

We have solved the B2BLOP with the schedules obtained from real instances of previous editions (tic-2012a, tic-2013a and forum-2013a), in order to compare the obtained results to the handmade ones, and with the schedule proposed for this year's edition of the forum. This last schedule, which we refer to as forum-2014a-t, has been obtained by the human expert by retracting 6 cancelled meetings, and by adding 15 last arrival meetings, to the schedule obtained for forum-2014a with the WPM1 method, in approximately 2.5 hours. The resulting B2BLOP instances have been solved with the WPM1 method in approximately 3, 120, 420 and 480 seconds respectively.

In Table 2 we provide a comparison on the quality of the solutions with respect to the number of idle time periods and location changes, when solved by hand and with WSimply.

Table 2. Number of idle time periods and location changes for the real instances when solved by hand and with WSimply. The maximum difference (homogeneity) between those numbers for distinct participants is given between parentheses.

Instance	# idle periods		# location changes	
	Handmade	WSimply	Handmade	WSimply
tic-2012a	20 (4)	0 (0)	112 (7)	103 (3)
tic-2013a	39 (4)	0 (0)	191 (9)	156 (4)
forum-2013a	99 (5)	0 (0)	27 (5)	105 (4)
forum-2014a-t		22 (2)		249 (6)

Note that the number of idle time periods is reduced to 0 when solving the problem with WSimply in almost all cases. In spite of this, we are still able to reduce the number of location changes with respect to the handmade solutions, except for the case of forum-2013a. But the solution obtained with WSimply in this case is still significantly better than the handmade one, which implied 99 idle time periods and was far less homogeneous. In any case, we are prioritizing the minimization of idle time periods of participants and the fairness of solutions, and leave location change minimization as a secondary desirable property.

5 Conclusion

In this work we have provided two distinct models for the B2BSOP and compared the efficiency of several types of solvers when dealing with some industrial and crafted instances of this problem, in a 'model-and-solve' approach. We also provide several new nontrivial industrial timetabling instances to the community.

The solutions found have been highly satisfactory for the human matchmaker, as they dramatically improve the handmade ones with respect to the number of idle time periods as well as location changes for the participants and, obviously, with quite less effort. Another aspect that the human matchmaker has really appreciated is the facility of adding meta-constraints to the model, like the ones we have used for achieving some level of fairness in the solutions. Fairness is crucial since participants may complain if they have the feeling of being discriminated, either with respect to idle time periods or with respect to location changes. The possibility of being able to fix partial solutions and adding new meetings to schedule has also been appreciated, since this is a hard task to do typically the day before the event due to last meeting request arrivals.

With respect to performance, we have noted that clasp is especially good on small instances, while WSimply (with the SBDD approach) appears to be better on bigger ones. However, it is not easy to draw conclusions, as we have not tuned the model for any solver in particular. The good results in the PB approach encourage us to develop a plain SAT model in the future, and to use MaxSAT solvers on this problem. Finally, it would be interesting to consider a handcrafted MIP model, and compare it against the MIP models obtained automatically from our models.

References

1. `http://www.b2match.com`. Accessed 11 Apr 2014.
2. `http://www.opturion.com`. Accessed 11 Apr 2014.
3. I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger. A New Look at BDDs for Pseudo-Boolean Constraints. *Journal of Artificial Intelligence Research (JAIR)*, 45:443–480, 2012.
4. T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
5. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. W-MiniZinc: A Proposal for Modeling Weighted CSPs with MiniZinc. In *Proceedings of the 1st International Workshop on MiniZinc (MZN 2011)*, 2011.
6. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Solving weighted CSPs with meta-constraints by reformulation into Satisfiability Modulo Theories. *Constraints*, 18(2):236–268, 2013.
7. C. Ansótegui, M. L. Bonet, and J. Levy. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *LNCS*, pages 427–440. Springer, 2009.
8. M. Bofill, J. Espasa, M. Palahí, and M. Villaret. An extension to Simply for solving Weighted Constraint Satisfaction Problems with Pseudo-Boolean Constraints. In *XII Spanish Conference on Programming and Computer Languages (PROLE 2012)*, pages 141–155, Almería, Spain, September 2012.
9. M. Bofill, M. Palahí, J. Suy, and M. Villaret. Boosting Weighted CSP Resolution with Shared BDDs. In *Proceedings of the 12th International Workshop on Constraint Modelling and Reformulation (ModRef 2013)*, pages 57–73, Uppsala, Sweden, September 2013.
10. B. Dutertre and L. de Moura. The Yices SMT solver. Tool paper available at `http://yices.csl.sri.com/tool-paper.pdf`, August 2006. Accessed 11 Apr 2014.
11. S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 16th Annual Symposium*, pages 184–193. IEEE, 1975.
12. M. Gebser, T. Glase, O. Sabuncu, and T. Schaub. Matchmaking with Answer Set Programming. In *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, volume 8148 of *LNCS*, pages 342–347. Springer, 2013.
13. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *clasp* : A Conflict-Driven Answer Set Solver. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, volume 4483 of *LNCS*, pages 260–265. Springer, 2007.
14. V. M. Manquinho, J. P. M. Silva, and J. Planes. Algorithms for Weighted Boolean Optimization. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *LNCS*, pages 495–508. Springer, 2009.
15. N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.
16. O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

17. C. Schulte, M. Lagerkvist, and G. Tack. Gecode. *Software download and online material at the website:* `http://www.gecode.org`, 2006. Accessed 11 Apr 2014.