Journal of Artificial Intelligence Research 74 (2022) 263-301 Submitted 01/2021; published 05/2022

Constraint Solving Approaches to the Business-to-Business Meeting Scheduling Problem

Miquel Bofill MBOFILL@IMAE.UDG.EDU
IMAE department, University of Girona, Girona, Spain

Jordi Coll JORDI.COLL@LIS-LAB.FR
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Marc Garcia MARCGOLIVERAS@GMAIL.COM
IMAE department, University of Girona, Girona, Spain

Jesús Giráldez-Cru JGIRALDEZ@UGR.ES
DaSCI institute, DECSAI, University of Granada, Granada, Spain

Gilles Pesant GILLES.PESANT@POLYMTL.CA
Polytechnique Montréal, Montréal, Canada

Josep Suy SUY@IMAE.UDG.EDU
Mateu Villaret VILLARET@IMAE.UDG.EDU
IMAE department, University of Girona, Girona, Spain

Abstract

The Business-to-Business Meeting Scheduling problem consists of scheduling a set of meetings between given pairs of participants to an event, while taking into account participants' availability and accommodation capacity. A crucial aspect of this problem is that breaks in participants' schedules should be avoided as much as possible. It constitutes a challenging combinatorial problem that needs to be solved for many real world brokerage events.

In this paper we present a comparative study of Constraint Programming (CP), Mixed-Integer Programming (MIP) and Maximum Satisfiability (MaxSAT) approaches to this problem. The CP approach relies on using global constraints and has been implemented in MiniZinc to be able to compare CP, Lazy Clause Generation and MIP as solving technologies in this setting. We also present a pure MIP encoding. Finally, an alternative viewpoint is considered under MaxSAT, showing best performance when considering some implied constraints. Experiments conducted on real world instances, as well as on crafted ones, show that the MaxSAT approach is the one with the best performance for this problem, exhibiting better solving times, sometimes even orders of magnitude smaller than CP and MIP.

1. Introduction

Business-to-business (B2B) events involve holding meetings between attendees with common interests or needs. These events typically occur in forums, conferences, and gatherings as an opportunity for the attendees to find investors, sell or buy products, share ideas and projects, etc. Obtaining a feasible schedule for the desired meetings according to time availability of participants and accommodation capacity is a hard combinatorial problem.

It is frequent that the participants of such events 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 pairwise meetings with other participants, with a certain priority. They also indicate their time availability for the meetings (notice that, in addition to the meetings, there are usually other activities running in parallel that participants may need to attend and therefore they should reserve some time slots). Moreover, the participants may ask for meetings in particular session, e.g. morning or afternoon sessions. Then, according to this information (participants' availability and priorities for the desired meetings) a matchmaker proposes a set of matches (meetings) to be scheduled. Scheduling these meetings is a tough task since the timetable must satisfy several hard constraints, like avoiding meeting collisions, as well as some soft constraints, like avoiding unnecessary breaks between meetings of the same participant. Experience shows that breaks may lead some participants to leave the event, dismissing later scheduled meetings. Therefore, it is desirable to avoid unnecessary breaks in the particular schedules of each participant. At the same time, it is desirable to be fair by avoiding big differences in the number of breaks of participants.

Several works have dealt with this problem. Gebser, Glase, Sabuncu, and Schaub (2013) describe an answer set programming system, being used by the company *piranha womex AG* for computing matchmaking schedules in several fairs. This system has some limitations. 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 multiple participants. *Model-and-solve* approaches on a more complete formulation have been developed in recent works. Several CP, MIP and SAT based encodings have been studied by Bofill, Espasa, Garcia, Palahí, Suy, and Villaret (2014), by Pesant, Rix, and Rousseau (2015) and by Bofill, Garcia, Suy, and Villaret (2015).

In this work we revisit, extend and improve the state-of-the-art approaches for the Business-to-Business Meeting Scheduling problem introduced by Pesant et al. (2015) and by Bofill et al. (2015), and compare their performance on real world instances, as well as on crafted instances. In particular:

- We consider some extensions of the problem: including time restrictions for meetings, meeting precedences, and prefixed meetings. Using these extensions, we contribute 180 new crafted instances.[1]
- We re-implement the Constraint Programming model by Pesant et al. (2015) using MiniZinc (Nethercote, Stuckey, Becket, Brand, Duck, & Tack, 2007), allowing us to compare the performance of different solving technologies using the same model.
- We provide an alternative way of identifying the breaks in participants' schedules.
- We provide further details of the MaxSAT encoding by Bofill et al. (2015).
- We improve the MaxSAT and MIP models by taking advantage of implied constraints occurring in the problem.

1. The instances are available at http://imae.udg.edu/recerca/lai.

- We adapt the Constraint Programming, MaxSAT and MIP models to deal with the extensions of the problem.

The rest of the paper is structured as follows. In Section 2 we define the problem at hand. In Sections 3, 4 and 5 we present the different models considered, namely Constraint Programming, MaxSAT and MIP models respectively. In Section 6 we provide a comparison of the presented models. Section 7 is devoted to experimental evaluation. A summary and conclusions are given in Section 8. Finally, we provide a detailed MiniZinc model in an appendix section.

2. The Business-to-Business Meeting Scheduling Problem

In this section we define the problem at hand, as well as some extensions of it.

Definition 2.1. *Let P be a set of participants to an event, T a set of available time slots and L a set of available locations for holding meetings. Let M be a set of unordered pairs of participants in P, representing the meetings to be scheduled.*

A schedule S is a total mapping from M to $T \times L$. Given a meeting m, by $time(S, m)$ and $loc(S, m)$ we refer to the time slot and the location assigned to m, respectively, according to S. In other words, given a meeting m with $S(m) = (t, l)$ we have $time(S, m) = t$ and $loc(S, m) = l$.

We define a feasible business-to-business (B2B) meeting schedule *S as 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.*

$$time(S, m_1) = time(S, m_2) \implies m_1 \cap m_2 = \emptyset \qquad m_1, m_2 \in M : m_1 \neq m_2 \quad (1)$$

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

$$time(S, m_1) = time(S, m_2) \implies loc(S, m_1) \neq loc(S, m_2) \qquad m_1, m_2 \in M : m_1 \neq m_2 \quad (2)$$

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

The B2BSP is NP-complete. It is clearly in NP, and its NP-hardness can be proved by reduction from the *edge coloring problem* (deciding if, given a graph G, all its edges can be colored so that no two incident edges have the same color, using k colors or the fewest number of colors, which amounts to the maximal degree of the graph).

Theorem 2.1. *The B2BSP is NP-hard.*

Proof. (sketch). The edge coloring problem can be reduced to the B2BSP as follows. Each vertex of the graph represents a participant to the B2B event, while each edge corresponds to a meeting between the two participants at the vertices. We take k as the number of time slots. We can take as many locations as needed. □

Alternatively, NP-hardness of the B2BSP can be shown by a reduction from the *restricted timetable problem* described by Even, Itai, and Shamir (1975).

Typically, we are interested in schedules which minimize the number of breaks. By a *break* 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. Without loss of generality we assume that time slots are consecutively numbered, starting at one.

Definition 2.2. *Given a B2B meeting 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:*

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

By $L_S(p)[i]$ we refer to the i-th element of $L_S(p)$, i.e., m_i, and $\exists!$ stands for exists unique.

Definition 2.3. *We define $H_S(p)$ as the number of breaks (or holes) of participant p for a given schedule S as follows:*

$$H_S(p) = |\{L_S(p)[i] \mid i \in 1..|L_S(p)| - 1, time(S, L_S(p)[i]) + 1 \neq time(S, L_S(p)[i+1])\}|$$

Definition 2.4. *We define the* B2B Scheduling Optimization Problem (B2BSOP) *as the problem of finding a feasible B2B meeting schedule S, where the total number of breaks of the participants is minimal, i.e., such that it minimizes*

$$\sum_{p \in P} H_S(p) \tag{3}$$

Depending on participants' requirements, more constraints can be imposed on the meetings:

- **Fixed sessions.** As an additional constraint, we consider the case where the B2B meetings can be structured in sessions, each one consisting of a set of consecutive time slots. Then, it may be the case that some meetings must necessarily be held in a particular session. Let's then consider that the set of time slots T is divided into disjoint sets $T_1, \ldots, T_s$ and, moreover, we have a mapping $session : M \to \{0, 1, \ldots, s\}$, where $session(m) = i \in \{1..s\}$ means that meeting m must take place at some time slot of session T_i, and $session(m) = 0$ means that it does not matter in which session meeting m is held. Then the schedule should also satisfy the following requirement:

$$(session(m) = i \wedge i > 0) \Longrightarrow time(S, m) \in T_i \qquad m \in M$$

- **Fixed Meetings.** A meeting can be requested to be held in a specific time slot. This would mean to have a mapping $fixed : M \to \{0\} \cup T$, with $fixed(m) = i \in T$ meaning

that meeting m must take place at time slot i, and $\mathit{fixed}(m) = 0$ meaning that it does not matter in which time slot meeting m takes place. Then the schedule should also satisfy the following requirement:

$$\mathit{fixed}(m) > 0 \Longrightarrow \mathit{time}(S, m) = \mathit{fixed}(m) \qquad m \in M$$

- **Forbidden time slots.** A participant could request not to have any meeting in some time slots. Then, we would have a mapping $\mathit{forb} : P \to 2^T$, with $\mathit{forb}(p)$ denoting the set of forbidden time slots for participant p. Then the schedule should also satisfy the following requirement:

$$p \in m \implies \mathit{time}(S, m) \notin \mathit{forb}(p) \qquad p \in P, m \in M$$

- **Meeting precedences.** Sometimes it may be required that some meetings are scheduled before others. Therefore, we would have a mapping $\mathit{prec} : M \to 2^M$, with $\mathit{prec}(m)$ denoting the set of meetings that must be held before m. Then the schedule should satisfy the following requirement:

$$\mathit{time}(S, m') < \mathit{time}(S, m) \qquad m \in M, m' \in \mathit{prec}(m)$$

Sometimes optimality with respect to the total number of breaks may not be enough, and a certain notion of fairness may be required:

- **Fairness.** We could consider, as a sort of meta-constraint, a fairness requirement on the number of breaks among participants. In particular, we could ask for feasible schedules S such that the difference between the number of breaks of every two distinct participants is bounded by a certain quantity d.

$$|H_S(p_1) - H_S(p_2)| \leq d \qquad p_1, p_2 \in P : p_1 \neq p_2$$

Note that feasibility and optimality can be guaranteed without knowing the exact location where each meeting will be held, but bounding the number of meetings to be held in the same time slot to the number of available locations. Therefore, in this work, we will not consider the location assigned to each meeting, but the total number of meetings per time slot.

Without loss of generality we restrict the models and experiments to two distinct session types, namely T_{AM} and T_{PM}, corresponding to morning and afternoon time slots, respectively, and such that $T = T_{AM} \uplus T_{PM}$.

For the sake of readability we define M_p to denote the set of meetings of participant p, and M_{AM} (resp. M_{PM}) to denote the set of meetings to be held in the morning (resp. afternoon) sessions.

3. Constraint Programming Models

A natural CP model for the B2BSOP defines variables

$$s_{pt} \qquad p \in P, t \in T$$

with domain in $M_p \cup \{0\}$ to represent which meeting participant p will hold at time t, value 0 corresponding to no meeting. Pesant et al. (2015) shown that the s_{pt} viewpoint was much better suited to CP than the alternative by Bofill et al. (2014). Therefore, in this section we consider this viewpoint, which is moreover very simple and essentially requires two kinds of global constraints:

- The *Global Cardinality Constraint*, that counts the number of occurrences of values in a sequence. This constraint will deal with the feasibility of the problem by ensuring that all meetings are scheduled, and that the available number of locations is not surpassed by the number of meetings scheduled at each time slot. Notice that the viewpoint itself ensures that a participant will have no two meetings scheduled at the same time.

- The *Cost Regular Constraint*, that forces a sequence of variables to be a word of a regular language and can assign costs to certain patterns. This constraint will be used to deal with the optimization of the schedule, by counting the number of breaks of the participants.

3.1 Feasibility

Here is our (abstract) model for the feasibility subproblem:

$$\texttt{gcc}(\{s_{p\star}\},\ \langle 0\rangle{+}{+}list(M_p),\ \langle\{|T|-|M_p|\},\{1\},\ldots,\{1\}\rangle) \qquad p \in P \quad (4)$$

$$\texttt{gcc}(\{s_{\star t}\},\ \langle 0\rangle{+}{+}list(M),\ \langle\{|P|-2|L|,\ldots,|P|\},\{0,2\},\ldots,\{0,2\}\rangle) \qquad t \in T \quad (5)$$

$$s_{pt} \in M_p \cup \{0\} \qquad p \in P,\ t \in T \quad (6)$$

where we use $list(X)$ to denote a permutation of the elements of set X, and we use $+\!+$ to denote list concatenation.

Constraints (4) use a global cardinality constraint (Régin, 1996) on the decision variables of a given participant to ensure that each of her meetings appears once (the first component of the vector of occurrences, corresponding to value 0, indicates the number of time slots without a meeting). Constraints (5) use a global cardinality constraint on the decision variables of a given time slot to express two things: the first component says that the number of participants not having a meeting must be at least $|P| - 2|L|$ because we can hold at most $|L|$ meetings and each meeting appears twice (once for each participant); the other components, for each meeting, say that the two participants to a given meeting must attend it in the same time slot and therefore a meeting occurs twice or not at all.

3.2 Optimization

We now model the optimization component of our problem by modelling break patterns. We define a variable b_p for each participant p giving the number of breaks in her schedule and seek to minimize the total number of breaks in the schedule. In order to link the b_p variables to the main s_{pt} variables we need to consider the sequence of values taken by the decision variables of a participant: each subsequence of zeros in between scheduled meetings for p corresponds to a break and b_p represents how many such breaks there are in the sequence. For example, patterns $0\star\star00\star00$ and $\star000\star0\star0$ for eight time slots and three meetings feature respectively one and two breaks.

To express this globally we could enumerate each possible pattern, associate its number of breaks and use a `table` constraint. For given T, M_p, and maximum number of breaks b' this makes

$$\sum_{i=0}^{b'} \binom{|M_p|-1}{i} \cdot \binom{|T|-|M_p|+1}{i+1}$$

patterns. Even if we restrict ourselves to at most $b' = 2$ breaks, the number of patterns is in $\Theta(|M_p|^2(|T|-|M_p|)^3)$ which, when the number of meetings is about half of the number of time slots, simplifies to $\Theta(|M_p|^5)$. Considering that the largest instance has 22 time slots with some participants holding 11 meetings, we could end up generating hundreds of thousands of patterns.

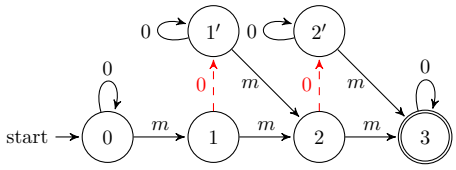

Figure 1: Automaton $\mathcal{A}_1$ for a participant with three meetings. Arc label "m" stands for any meeting and label "0" for no meeting. Only the red dashed arcs carry a cost, of one unit, to mark the start of a break.

A much more compact way to express this uses an automaton on $2|M_p|$ states that recognizes precisely these patterns. Figure 1 presents such an automaton, referred to as $\mathcal{A}_1$, for a participant with three meetings. Observe however that by concentrating on patterns without distinguishing between meetings we may miss some inferences. For example any assignment from the sequence of domains $\langle\{m_1, m_2, m_3, 0\}, \{m_1, m_2, m_3, 0\}, \{m_4, 0\}, \{m_4, 0\}, \{m_1, m_2, m_3, 0\}, \{m_1, m_2, m_3, 0\}\rangle$ corresponding to four meetings being scheduled over six time slots (some of them forbidden for some meetings) will necessarily introduce at least one break but such an automaton will not recognize it. To catch this, a more fine-grain automaton distinguishing between meetings will have $2^{|M_p|+1} - 2$ states essentially representing all subsets of meetings. Figure 2 presents such an automaton, referred to as $\mathcal{A}_2$, again for a participant with three meetings. Because this automaton has significantly more states, we will refrain from using it.

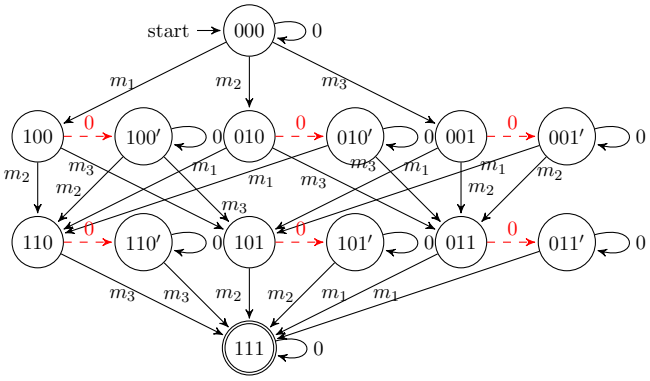

Figure 2: Automaton $\mathcal{A}_2$ for a participant with three meetings. Arc label "m_i" stands for that particular meeting and label "0" for no meeting. Only the red dashed arcs carry a cost, of one unit, to mark the start of a break.

Notice that, in fact, both automata are only accepting sequences with the exact number of meetings of each participant. However, Constraints (4) are already enforcing that all meetings of each participant are scheduled exactly once. Therefore, we can use an even more compact automaton that is only taking care of the break patterns. Figure 3 presents such automaton that we refer to as $\mathcal{A}_0$. This automaton has only three states and does not care about the number of meetings occurring in the sequence, so it will be the same for each participant.

When using automaton $\mathcal{A}_1$, Constraints (4) are still needed because the automaton is only counting the number of meetings of each participant but does not control at all if there is any repetition. Conversely, when using automaton $\mathcal{A}_2$ we could save the use of Constraints (4) because this automaton only accepts participants' schedules with all their meetings occurring exactly once.

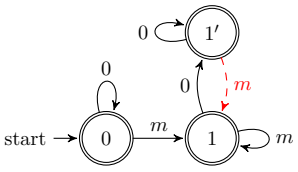

Figure 3: Automaton $\mathcal{A}_0$ for a participant with any number of meetings. Arc label "m" stands for any meeting and label "0" for no meeting. Only the red dashed arc carries a cost, of one unit, to mark the end of a break.

Then, the objective function and related constraints are defined as follows:

$$\min \sum_{p \in P} b_p \qquad \text{s.t.} \tag{7}$$

$$b_p = 0 \qquad p \in P : \ |M_p| \in \{0, 1, |T|\} \tag{8}$$

$$\texttt{cost\_regular}(\langle s_{p\star} \rangle, \mathcal{A}, b_p) \qquad p \in P : \ 1 < |M_p| < |T| \tag{9}$$

$$b_p \in \mathbb{N} \qquad p \in P \tag{10}$$

Constraints (8) fix b_p to zero for participants who trivially have no break in their schedule (e.g., they have a single meeting or as many meetings as there are time slots).

The `cost_regular` constraint (9) on any of the previously given automata $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_0)$ makes variable b_p equal to the sum of the costs of the arcs on the path corresponding to the values taken by the sequence of variables $\langle s_{p\star} \rangle$ (see Demassey, Pesant, & Rousseau, 2006). An upper bound on b_p will limit the feasible paths in the automaton and possibly remove arcs (i.e., filter values in a domain) that do not belong to any feasible path. Conversely, the smallest cost of the possible paths given the current domains of the variables provides a lower bound on b_p.

Regarding non-zero labels of automata $\mathcal{A}_1$ and $\mathcal{A}_0$, they can be meeting identifiers or they can be a reified value representing that there is a scheduled meeting or not. This second option requires the introduction of reification variables but reduces the size of the automaton since the number of possible transitions per state is just two (see Appendix A for more implementation details).

3.3 Additional Constraints

We now describe the constraints that need to be added for the variations of the problem described in Section 2.

- **Fixed sessions.** Constraints (11) and (12) disallow afternoon (resp. morning) meetings taking place during morning (resp. afternoon) time slots.

$$s_{pt} \in M_{AM} \cup \{0\} \qquad p \in P, t \in T_{AM} \tag{11}$$

$$s_{pt} \in M_{PM} \cup \{0\} \qquad p \in P, t \in T_{PM} \tag{12}$$

- **Fixed meetings.** Constraints (13) ensure that every fixed meeting is scheduled in its corresponding time slot for each of its two participants. Recall that the fact that each meeting is scheduled exactly once is already enforced by global cardinality constraints (4) or the automata.

$$\begin{aligned} s_{p_1, \mathit{fixed}(m)} &= m \\ s_{p_2, \mathit{fixed}(m)} &= m \end{aligned} \qquad m = (p_1, p_2) \in M : \mathit{fixed}(m) > 0 \tag{13}$$

- **Forbidden time slots.** Constraints (14) ensure that there is no meeting scheduled for a participant during one of his forbidden time slots.

$$s_{pt} = 0 \qquad p \in P, t \in \mathit{forb}(p) \tag{14}$$

- **Meeting precedences.** Constraints (15) define auxiliary variables indicating the time slot in which each meeting is held, which are linked, using an implicit *element* constraint, to the main decision variables through Constraints (16). Constraints (17) then express the precedences between meetings.

$$t_m \in T \qquad m \in M \tag{15}$$
$$s_{pt_m} = m \qquad p \in P, m \in M_p \tag{16}$$
$$t_{m'} < t_m \qquad m \in M, m' \in prec(m) \tag{17}$$

- **Fairness.** Constraints (18) ensure some fairness between individual schedules by requiring that the number of breaks among individual schedules differ by at most some integer parameter d.

$$b_p - \min_{p' \in P} b_{p'} \leq d \qquad p \in P \tag{18}$$

4. MaxSAT Models

In this section we present the MaxSAT formulations for the B2BSOP, starting with the base encoding, which is next extended in order to improve its performance in practice. We describe a meta-encoding using cardinality constraints such as *at-most-k* or *exactly-k*. These constraints can be translated into MaxSAT in different ways. In Section 4.6 we detail the chosen method for each of these constraints.

4.1 MaxSAT Base Encoding

The B2BSOP can be encoded to a partial MaxSAT formula (Li & Manyà, 2009), where some clauses are marked as hard whereas others are marked as soft, and the goal is to find an assignment to the variables that satisfies all hard clauses and falsifies the minimum number of soft clauses. In our case, the falsification of a soft clause will represent the existence of a break for some participant.

- **Variables and viewpoint.** The MaxSAT model is based in the viewpoint given by Boolean variables stating whether a meeting is scheduled or not in a time slot. With these variables we are able to model feasibility, and all additional constraints not related to breaks (i.e., not related to fairness nor optimization). The variables are

$$schedule_{i,j} : \text{meeting } i \text{ is held at time slot } j$$

- **Basic constraints for feasibility.**
  - At most one meeting involving the same participant is scheduled at each time slot.

$$atMost(1, \{schedule_{i,j} \mid i \in M_p\}) \qquad p \in P, j \in T \tag{19}$$

  - Each meeting not in a fixed session is scheduled in a time slot (see also (22) and (24)).

$$exactly(1, \{schedule_{i,j} \mid j \in T\}) \qquad i \in M \setminus (M_{AM} \cup M_{PM}) \tag{20}$$

– There are at most as many meetings scheduled in a given time slot as available locations $|L|$.

$$atMost(|L|, \{schedule_{i,j} \mid i \in M\}) \qquad j \in T \tag{21}$$

With these constraints we get a total mapping from meetings to time slots, except for those meetings that must be held in a particular session (M_{AM} and M_{PM} meetings). Those are considered in the following subsection.

4.2 Additional Constraints

We now describe the constraints that need to be added for the variations of the problem described in Section 2.

- **Fixed sessions.** Each meeting with a fixed session is scheduled in a slot of the required session, and is not scheduled in a slot of another session.

$$exactly(1, \{schedule_{i,j} \mid j \in T_{AM}\}) \qquad i \in M_{AM} \tag{22}$$

$$\neg schedule_{i,j} \qquad i \in M_{AM}, j \in T_{PM} \tag{23}$$

$$exactly(1, \{schedule_{i,j} \mid j \in T_{PM}\}) \qquad i \in M_{PM} \tag{24}$$

$$\neg schedule_{i,j} \qquad i \in M_{PM}, j \in T_{AM} \tag{25}$$

- **Fixed meetings.** Every fixed meeting is scheduled in its corresponding time slot.

$$schedule_{m,fixed(m)} \qquad m \in M : fixed(m) > 0 \tag{26}$$

- **Forbidden time slots.** No meeting is scheduled in a forbidden time slot for any of its participants.

$$\bigwedge_{i \in M_p,\, j \in forb(p)} \neg schedule_{i,j} \qquad p \in P \tag{27}$$

- **Meeting precedences.** All meetings are scheduled after the meetings that must precede them.

$$schedule_{i',j'} \rightarrow \neg schedule_{i,j} \qquad i \in M, i' \in prec(i), j, j' \in T : j' \geq j \tag{28}$$

- **Fairness.** In order to be able to enforce fairness among participants' schedules, as to minimize the overall number of breaks, we need to be able to detect idle time slots. To this end, we introduce some auxiliary variables:

$usedSlot_{p,j}$: participant p has a meeting scheduled at time slot j.
$meetingHeld_{p,j}$: participant p has a meeting scheduled at or before time slot j.
$endHole_{p,j}$: participant p has an idle time period (break) finishing at time slot j.

We also need to add constraints to define the semantics of these variables:

– If a meeting is scheduled in a certain time slot, then that time slot is used by both participants of the meeting.

$$schedule_{i,j} \rightarrow (usedSlot_{p_1,j} \wedge usedSlot_{p_2,j}) \qquad i = (p_1, p_2) \in M, j \in T \tag{29}$$

In the reverse direction, if a time slot is used by some participant, then one of the meetings of that participant must be scheduled at that time slot.

$$usedSlot_{p,j} \rightarrow \bigvee_{i \in M_p} schedule_{i,j} \qquad p \in P, j \in T \tag{30}$$

– For each participant p and time slot j, $meetingHeld_{p,j}$ is true if and only if participant p has had a meeting at or before time slot j.

$$\neg usedSlot_{p,1} \rightarrow \neg meetingHeld_{p,1} \quad p \in P \tag{31}$$

$$(\neg meetingHeld_{p,j-1} \wedge \neg usedSlot_{p,j}) \rightarrow \neg meetingHeld_{p,j} \quad p \in P, j \in T \setminus \{1\} \tag{32}$$

$$usedSlot_{p,j} \rightarrow meetingHeld_{p,j} \qquad p \in P, j \in T \tag{33}$$

$$meetingHeld_{p,j-1} \rightarrow meetingHeld_{p,j} \qquad p \in P, j \in T \setminus \{1\} \tag{34}$$

Now we can characterize breaks in participants' schedules. If some participant does not have any meeting in a certain time slot, but she has had some meeting before, then she is having break. We reify such pattern with auxiliary variables $endHole_{p,j}$ in order to count the number of breaks of each participant. This will allow us to find the maximum and minimum number of breaks among all participants, and to enforce fairness by bounding their difference.

– $endHole_{p,j}$ is true if and only if participant p has a break finishing at time slot j.

$$endHole_{p,j} \leftrightarrow \neg usedSlot_{p,j} \wedge meetingHeld_{p,j} \wedge usedSlot_{p,j+1} \quad p \in P, j \in T \setminus \{|T|\} \tag{35}$$

– The list of variables $sortedHole_{p,1}, \ldots, sortedHole_{p,|T|-1}$ corresponds to the unary representation of the number of breaks of each participant p. In other words, $sortedHole_{p,j}$ holds iff participant p has at least j breaks in her schedule. This list is obtained by sorting decreasingly the *endHole* variables.

$$sort([endHole_{p,j} : j \in T], [sortedHole_{p,j} : j \in T \setminus \{|T|\}]) \qquad p \in P \tag{36}$$

– $max_1, \ldots, max_{\lfloor(|T|-1)/2\rfloor}$ and $min_1, \ldots, min_{\lfloor(|T|-1)/2\rfloor}$ are unary representations of bounds on the maximum and minimum number of breaks among all participants, respectively.

$$sortedHole_{p,j} \rightarrow max_j \qquad p \in P, j \in 1..\lfloor(|T|-1)/2\rfloor \tag{37}$$
$$\neg sortedHole_{p,j} \rightarrow \neg min_j \qquad p \in P, j \in 1..\lfloor(|T|-1)/2\rfloor \tag{38}$$

Note that there can be at most $\lfloor(|T|-1)/2\rfloor$ breaks per participant.

Constraints (37) ensure that $max_1,\ldots,max_{\lfloor(|T|-1)/2\rfloor}$ (interpreted as a unary representation of a number) is greater or equal than the maximum number of holes among all participants, and Constraints (38) ensure that $min_1,\ldots,min_{\lfloor(|T|-1)/2\rfloor}$ is smaller or equal than the minimum number of holes among all participants.

Finally, Constraints (39) and (40) enforce the required fairness degree.

– The difference between the maximum and minimum number of breaks must be at most d.

$$\neg min_j \wedge max_j \rightarrow dif_j \qquad j \in 1..\lfloor(|T|-1)/2\rfloor \tag{39}$$
$$atMost(d, \{dif_j \mid j \in 1..\lfloor(|T|-1)/2\rfloor\}) \tag{40}$$

4.3 Optimization

Minimization of the number of breaks is achieved by means of soft constraints. We simply post as soft constraint the negation $sortedHole_{p,j}$ variables. Knowing that each participant will have at most $\lfloor(|T|-1)/2\rfloor$ breaks, we have a limited number of soft constraints.

$$\neg sortedHole_{p,j} \qquad p \in P, j \in 1..\lfloor(|T|-1)/2\rfloor \tag{41}$$

Note that, each $sortedHole_{p,j}$ variable set to true, increases the cost by 1.

In case there is no homogeneity (i.e., $d = \infty$) we could simply post as soft constraints the reified clauses characterizing holes from Constraints (35).

$$(\neg usedSlot_{p,j} \wedge meetingHeld_{p,j}) \rightarrow \neg usedSlot_{p,j+1} \quad p \in P, j \in T \setminus \{|T|\} \tag{42}$$

Remark 4.1. *If we were considering optimization but no fairness, Constraints* (31) *and* (32) *would not be necessary, since minimization of the number of breaks would force the value of* $meetingHeld_{p,j}$ *to be false for every participant* p *and time slot* j *previous to the first meeting of* p*. However, since we are also seeking fairness, these constraints are mandatory. Without them, the value of* $meetingHeld_{p,j}$ *could be set to true for time slots* j *previous to the first meeting of* p*, inducing a fake break in order to satisfy the (hard) fairness constraints defined in Section 4.2.*

4.4 Implied Constraints

We have identified the following implied constraints.

- **Implied Constraint 1.** The number of meetings of a participant p as derived from $usedSlot_{p,j}$ variables must match the total number of meetings of p.

$$exactly(|M_p|, \{usedSlot_{p,j} \mid j \in T\}) \qquad p \in P \tag{43}$$

- **Implied Constraint 2.** The number of participants having a meeting in a given time slot is bounded by twice the number of available locations.

$$atMost(2 \times |L|, \{usedSlot_{p,j} \mid p \in P\}) \qquad j \in T \tag{44}$$

As we will see in the experiments, both implied constraints are really helpful and, in combination, they allow to solve all original instances within the given time limit. A possible reason for such efficiency is that they are able to detect inconsistent partial assignments that are not detectable without these implied constraints. Consider for instance a participant p with still m meetings to schedule within m still available time slots. Suppose that one of such time slots, say j, became unavailable due to the scheduling of as many meetings as available locations, and that none of these meetings involves participant p. Then, Constraint (44) will set variable $usedSlot_{p,j}$ to false but this will clash with Constraint (43) that will enforce $usedSlot_{p,j}$ to be true.

4.5 Further Improvements

We have also improved the encoding taking into account further knowledge of the problem. Namely, we know that the number of participants having a meeting in a given time slot is even because meetings always involve two distinct participants. We also know that among all the meetings of a same participant, just one can take place in a particular timeslot.

4.5.1 Even number of participants

Let $o_{j,1}, ...o_{j,2\times|L|}$ be the output variables of the cardinality network corresponding to Constraint (44) for a given timeslot $j \in T$. To enforce that the number of participants is an even number in all time slots we add the following clauses:

$$o_{j,i} \rightarrow o_{j,i+1} \qquad i \in \{1, 3, ...2 \times |L| - 1\}, j \in T \tag{45}$$

4.5.2 At most one meeting of a same participant

When counting the number of scheduled meetings in a timeslot, in Constraints (21), we have to bound them by the number of available locations $|L|$. However, we do not need to consider all possible meetings simultaneously because plenty of them are mutually exclusive, e.g., among all meetings of the same participant just one can be held.

Therefore, instead of considering all meetings in Constraints (21) we will consider clusters of meetings, where all meetings of a cluster share the same participant, hence being mutually exclusive. We greedily compute a partition Π of meetings where each subset is one of such clusters. Then, for each cluster c and timeslot $j \in T$ we introduce an auxiliary variable $cl_{c,j}$. We add the following clauses:

$$schedule_{i,j} \rightarrow cl_{c,j} \qquad c \in \Pi, i \in c, j \in T \tag{46}$$

and change Constraints (21) by the following:

$$atMost(|L|, \{cl_{c,j} \mid c \in \Pi\}) \qquad j \in T \tag{47}$$

The idea is that since at most one meeting of each cluster can be scheduled in each time slot, it suffices to limit the number of active clusters to $|L|$ at any time. This results in a significant reduction on the number of variables and clauses of the generated formulas. This is so because the number of input variables of the $atMost$ constraints of Constraints (21) are reduced between 2 and 4 times. For example, the input of those cardinality constraints for instance ticf-14crafa is reduced from 302 to just 79 variables. This results in a formula of just 119,577 variables and 389,280 clauses, instead of the 241,149 variables and 632,820 clauses of the corresponding formula without this improvement. Moreover, as we will see in Table 3, solving time of this instance is almost halved.

4.6 Encoding of Global Constraints

The cardinality constraints stating that at most (*atMost*) or exactly (*exactly*) k of a given set of variables must be true have been encoded as follows.

- $atMost(1, \_)$: quadratic number of pairwise mutex clauses.
- $exactly(1, \_)$: commander-variable encoding (Klieber & Kwon, 2007).
- $exactly(k, \_)$: cardinality networks (Abío et al., 2013).
- $atMost(k, \_)$ of Constraints (21), (40) and (47): sequential counter (Sinz, 2005).
- $atMost(k, \_)$ of Constraints (44): cardinality network (Abío et al., 2013).
- $sort(\_, \_)$ of Constraints (36): cardinality network (Abío et al., 2013).

These are the encodings that we found to return the best results on this problem.

5. Mixed-Integer Programming Model

In this section we provide a native MIP formulation that exactly solves the B2BSOP. This formulation is based on the logical model presented by Pesant et al. (2015), extended with new constraints to encode additional participants' requirements. In this formulation, the feasibility of the problem is encoded with a set of binary variables representing that a meeting is held in a particular time slot, where a set of constraints on these variables enforces a feasible schedule without considering the minimization of breaks. Then, the objective function is encoded using binary variables determining the terminating time slots of the breaks of each participant, which are linked to the feasibility problem through a series of linearized logical constraints. Although there exist other MIP formulations to this problem (Pesant et al., 2015), they do not show remarkable improvements solving real-world B2B instances, so they are not considered in this work.

At the end of this section we also provide further improvements of the introduced MIP formulation.

5.1 Feasibility

We first give the formulation to enforce the feasibility of a solution. Let x_{mt} be a binary variable that equals 1 if a meeting m is held at time t. Using these variables, the following constraints enforce the feasibility of a schedule:

$$\sum_{t \in T} x_{mt} = 1 \qquad m \in M \tag{48}$$

$$\sum_{m \in M_p} x_{mt} \leq 1 \qquad p \in P, t \in T \tag{49}$$

$$\sum_{m \in M} x_{mt} \leq |L| \qquad t \in T \tag{50}$$

$$x_{mt} = 0 \qquad m \in M_{AM}, t \in T_{PM} \tag{51}$$

$$x_{mt} = 0 \qquad m \in M_{PM}, t \in T_{AM} \tag{52}$$

Constraints (48) force each meeting to be held exactly once. Constraints (49) force each participant to be in at most one meeting at a time. Constraints (50) force the number of meetings at any time to be at most the number of available locations. Finally, Constraints (51) and (52) force the fixed sessions.

5.2 Optimization

In order to define the optimization function of the B2BSOP, we define the following variables:

$b' \in \mathbb{N}$	Variable upper bounding the maximum number of breaks assigned to any participant.
$y_{pt} \in \{0,1\}$	Indicator if participant p has a meeting at time t.
$z_{pt} \in \{0,1\}$	Equals 1 for time t starting from participant p's first meeting.
$h_{pt} \in \{0,1\}$	Indicator if time t terminates a break for participant p.

The variables h_{pt} contribute a value of 1 to the objective function. The necessary constraints linking these variables to the model are the following.

$$\sum_{t \in T} y_{pt} = |M_p| \qquad p \in P \tag{53}$$

$$x_{mt} \leq y_{pt} \qquad p \in P, m \in M_p, t \in T \tag{54}$$

$$y_{pt} \leq z_{pt} \qquad p \in P, t \in T \tag{55}$$

$$z_{pt} \leq z_{p,t+1} \qquad p \in P, t \in T, t \leq |T| - 1 \tag{56}$$

$$y_{p,t+1} - h_{pt} \leq y_{pt} + 1 - z_{pt} \qquad p \in P, t \in T, t \leq |T| - 1 \tag{57}$$

$$\sum_{t \in T} h_{pt} \leq b' \qquad p \in P \tag{58}$$

$$\sum_{t \in T} h_{pt} \geq b' - d \qquad p \in P \tag{59}$$

Constraints (53) through (56) link the y and z variables to the formulation. Constraints (57) then link the h variables to the formulation, forcing a break to be counted after an idle period. Constraints (58) and (59) are fairness constraints that limit the difference in the number of breaks per participant. We then define the problem as the minimization of the objective function:

$$\sum_{p \in P} \sum_{t \in T} h_{pt} \tag{60}$$

subject to constraints (48) through (52) and (53) through (59).

5.3 Additional Constraints

We add the following constraints for the extensions of the problem described in Section 2 that are not supported by Pesant et al. (2015):

- **Fixed meetings.** Every fixed meeting is scheduled in its corresponding time slot.

$$x_{m,fixed(m)} = 1 \qquad m \in M : fixed(m) > 0 \tag{61}$$

- **Forbidden time slots.** No meeting is scheduled in a forbidden time slot for any participant.

$$y_{pt} = 0 \qquad p \in P, t \in forb(p) \tag{62}$$

- **Meeting precedences.** Every meeting is scheduled after the meeting that precedes it.

$$x_{mt} \leq \sum_{t' < t} x_{m't'} \qquad m \in M, m' \in prec(m), t \in T \tag{63}$$

5.4 Further Improvements

In order to improve the model, we also add the following implied constraints.

$$\sum_{p \in P} y_{pt} \leq 2 * |L| \qquad t \in T \tag{64}$$

$$z_{p(|T|)} = 0 \qquad p \in P : |M_p| = 0 \tag{65}$$

$$z_{p(|T|)} = 1 \qquad p \in P : |M_p| > 0 \tag{66}$$

$$z_{pt} = 1 \qquad p \in P, t \in T : |M_p| = |T| \tag{67}$$

$$h_{pt} = 0 \qquad p \in P, t \in T : |M_p| \leq 1 \vee |M_p| = |T| \tag{68}$$

$$d \geq b' - \max(0, \min_{p \in P}(\min(|M_p| - 1, |T| - |M_p|))) \tag{69}$$

Constraint (64) is analogous to Implied Constraint (44) of the MaxSAT model. Notice however, that Implied Constraint (43) of the MaxSAT model is analogous to Constraint (53) of the MIP model. Constraints (65), (66) and (67) enforce that variables z_{pt} have the right value in the last time slot, according to the number of meetings of each participant. Constraint (68) enforces the right value of variables h_{pt} for those participants that will not have any break according to their number of meetings. Finally, Constraint (69) bounds b' taking into account the maximum possible number of breaks of the participant.

6. Comparing Models

Even though the models described in the previous sections are meant to be used with different solving approaches and therefore are likely to each be expressed in the most appropriate yet distinct formalism for the approach used, it can be instructive to examine the main differences between them, beyond the usual observation that CP models typically use fewer variables (not being Boolean/binary) and fewer constraints (thanks to global constraints representing common combinatorial substructures).

The most immediate and fundamental difference is in the choice of variables. The CP models are *participant-centered*, using variables whose value represents the meeting attended by a participant at a given time. This makes it easy to retrieve what a given participant is doing at a particular time but not so much when a given meeting is set to happen. The MaxSAT models, following what was proposed by Bofill et al. (2014), are *meeting-centered*, featuring Boolean variables stating whether a meeting is happening at a given time. This makes it easy to retrieve when a given meeting occurs but not so much a given participant's schedule. The MIP model is *meeting-centered* as well and almost identical to the MaxSAT models as far as the feasibility part is concerned: Boolean variables become binary variables and cardinality constraints become (pseudo-Boolean) linear constraints.

The meeting-centered viewpoint could have been an alternative for CP however such a representation would not have allowed expressing constraints directly on the sequence of meetings for a participant, which is important to evaluate the cost of an individual schedule and ultimately the objective we seek to minimize. All constraints in the CP models are easily expressed using the participant-centered variables except if we add meeting precedences, for which auxiliary variables providing a meetings viewpoint are defined and linked to the main variables. Feasibility constraints in the MaxSAT and MIP models are easily expressed using meeting-centered variables but optimization and fairness constraints — both related to a participant's schedule — require auxiliary variables.

Another distinctive feature is how optimization is handled. For our CP and MIP approaches, all constraints are considered hard. In the CP model, optimization constraint `cost_regular` assigns a unit cost to each break in a schedule and provides a cost variable thus summing breaks and that can be used by the objective function. The logical MIP model described in this paper defines the objective function through auxiliary variables but other MIP models described in Pesant et al. (2015), closer to the CP model, linearize the `cost_regular` formulation. In contrast, for our MaxSAT approach some constraints — those enforcing that there should be no break — are specified as soft so that maximizing the number of satisfied soft constraints achieves the desired objective.

For all three approaches, sometimes adding redundant/implied constraints (cuts) helps solve problems faster. Such is the case here for the implied constraints mentioned for MaxSAT and MIP — these do not need to be added in our CP models because they are already expressed through the global cardinality constraints.

7. Evaluation

In this section we describe the results of the experimental investigation performed. In particular, we present:

Table 1: Description of original B2B instances.

feature	forum-13	forum-13crafb	forum-13crafc	forum-14	forumt-14	forumt-14crafc	forumt-14crafd	forumt-14crafe	tic-12	tic-12crafc	tic-13	tic-13crafb	tic-13crafc	tic-14crafa	tic-14crafc	tic-14crafd	ticf-13crafa	ticf-13crafb	ticf-13crafc	ticf-14crafa
#participants	70	76	70	78	60	60	60	60	42	42	47	46	47	60	60	60	70	76	70	78
#meetings	154	195	154	302	237	237	236	236	125	125	180	184	180	237	237	236	154	195	154	302
#time slots	21	22	21	22	10	10	10	10	8	8	10	10	10	10	10	10	21	22	21	22
#locations	14	14	12	22	27	25	25	24	21	16	21	21	19	27	25	25	14	14	12	22
%forbidden	6.3	4.6	6.3	10.8	12.3	12.3	12.3	12.3	12.5	12.5	10	10	10	10	10	10	4.8	4.6	4.8	4.6
AM/PM?	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y
%morningslots	13	12	13	12	-	-	-	-	-	-	-	-	-	-	-	-	13	12	13	12

- An evaluation of the different CP models, using $\mathcal{A}_0$ and $\mathcal{A}_1$ automata, considering the complete schedule of each participant or a reified version of it. This is done in Subsection 7.1.
- An evaluation of the different MaxSAT encodings with the distinct improvements proposed on the original instances. This is done in Subsection 7.2.
- An evaluation of the proposed MIP models, again on the original instances. This is done in Subsection 7.3.
- A comparison of the best CP, MaxSAT and MIP configurations. The comparison is done on the original instances as well as on crafted versions of those original instances where additional constraints have been considered (fixed sessions, fixed meetings, forbidden time slots, precedences and distinct fairness values). This is done in Subsection 7.4.

In Table 1 we describe the characteristics of the original B2B instances used in the experiments. In particular, we report the number of participants, meetings, time slots and locations for each instance, as well as the percentages of forbidden time slots and morning time slots. We recall that these instances have been already used in previous works (Bofill et al., 2014; Pesant et al., 2015; Bofill et al., 2015). There are no instances with fixed meetings or meeting precedences. As in previous works, the value of fairness d has been set to 2.

The experiments have been run on a cluster of CPU nodes Intel Xeon E3-1220v2 at 3.10 GHz with 8GB of RAM. The timeout is 7200 seconds in all experiments.

7.1 CP Evaluation

We have considered the models described in Section 3 with automata $\mathcal{A}_0$ and $\mathcal{A}_1$ (automaton $\mathcal{A}_2$ was too big). Moreover, we have used these automata either on the s variables (the ones describing the **complete** schedule of each participant) or on a reification of them (**reified**) meaning *having meeting/not having meeting*.

These models have been implemented using MiniZinc. Thanks to this we have been able to easily compare three solving methodologies: pure CP with Gecode[2] (Schulte, Tack, &

2. https://github.com/Gecode/gecode.

Lagerkvist, 2019) version 6.2.0, Lazy Clause Generation with Chuffed[3] (Chu, 2011) commit 23b9fce, and integer programming with IBM ILOG Cplex[4] (Cplex, 2009) version 12.9.0. In Appendix A we provide a complete MiniZinc model and details on how to build the different variants.

Regarding search strategies,[5] we considered `input_order` and `first_fail` strategies for s variables, resulting in no significant performance difference but being slightly better `input_order`. This is the search strategy used for Gecode. However, for Chuffed we obtained much better results with a search strategy that first tries to make the objective value as small as possible, and then continues with `input_order` over variables s. Moreover, for Chuffed we also enable the *free search* option, which alternates between the given search strategy and activity-based search strategy.

Table 2: Comparison of $\mathcal{A}_1$ and $\mathcal{A}_0$ with Chuffed solver. Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

	$\mathcal{A}_1$				$\mathcal{A}_0$			
instance	complete		reified		complete		reified	
forum-13	TO	1	1427.4	0	4182.5	0	2891.2	0
forum-13crafb	TO	26	1808.3	6	3551.0	6	1725.6	6
forum-13crafc	TO	20	TO	10	TO	21	TO	14
forum-14	TO	104	3465.8	2	3802.1	2	873.9	2
forumt-14	10.9	5	4.2	5	12.6	5	5.6	5
forumt-14crafc	72.5	5	29.0	5	35.8	5	20.2	5
forumt-14crafd	63.7	4	30.9	4	32.0	4	20.9	4
forumt-14crafe	56.3	5	37.5	5	34.9	5	26.3	5
tic-12	2.3	0	0.8	0	2.5	0	0.8	0
tic-12crafc	2.1	0	0.9	0	4.1	0	1.2	0
tic-13	23.6	0	17.3	0	45.6	0	16.9	0
tic-13crafb	8.2	0	3.2	0	4.1	0	2.0	0
tic-13crafc	1746.0	4	847.6	4	1643.9	4	518.2	4
tic-14crafa	54.0	0	25.0	0	46.4	0	22.4	0
tic-14crafc	44.8	0	34.4	0	55.3	0	24.1	0
tic-14crafd	11.5	0	3.4	0	21.4	0	6.2	0
ticf-13crafa	TO	2	5657.7	0	TO	2	2973.4	0
ticf-13crafb	TO	22	3026.2	3	3932.0	3	2291.9	3
ticf-13crafc	TO	20	TO	11	TO	21	TO	13
ticf-14crafa	TO	116	5249.8	0	TO	71	1714.8	0
#solved	12		18		16		18	

In Table 2 we report on the evaluation of the four possible configurations with Chuffed on the original instances. As we can observe, **reified** always solves more instances than

3. https://github.com/chuffed/chuffed.
4. https://www.ibm.com/products/ilog-cplex-optimization-studio.
5. The search strategy originally used by Pesant et al. (2015) is not available in MiniZinc.

complete. Also, $\mathcal{A}_0$ is better than $\mathcal{A}_1$, since it solves more instances in **complete**, and the same instances but generally faster in **reified**. This may well be due to the fact that $\mathcal{A}_0$ with reification results in the smallest configuration. Therefore, $\mathcal{A}_0$ and **reified** is the chosen configuration for the remaining experiments when using Chuffed. We have also tested Gecode and Cplex solvers with the four configurations of automata, which are not competitive as shown in Subsection 7.4.1. In particular, the results with Gecode are not competitive at all, but we include in Table 5 the results with $\mathcal{A}_0$ and **reified** which is the best option. Regarding Cplex, $\mathcal{A}_0$ and **reified** is also the best option, but as we will observe in Table 5, using the native MIP model gives much better results than using a MIP translation of the CP models.

7.2 MaxSAT Evaluation

In this section we analyze the efficiency of the MaxSAT encodings provided in Section 4, namely: basic (corresponding to the MaxSAT base encoding), imp1, imp2, imp12 and imp12$^+$, which correspond to the encodings that incorporate the first, the second, and both implied constraints, as well as the further improvements, respectively. The MaxSAT solver used is UWrMaxSAT[6] (Piotrów, 2019) version 1.0, which was one of the best MaxSAT solvers of the unweighted track of the 2019 MaxSAT Evaluation.

In Table 3, we report on the solving times obtained with each approach on the set of original instances. As we can see, adding implied constraints results in better performances. In particular, with imp12 we are able to solve all the 20 instances (in Section 4.4 we provide possible hints on the interaction between the implied constraint to early prune wrong partial assignments). Another interesting observation is that imp1 seems to be a bit faster than imp2, but each of these two encodings is able to solve two instances that cannot be solved by the other. This suggests that imp1 and imp2 may be somehow complementary in harder instances. Finally, imp12$^+$ is able to improve the solving times in 19 of the 20 instances considered with respect to imp12, hence this is the chosen encoding for the remaining experiments and we refer to it as MaxSAT.

7.3 MIP Evaluation

In this section we compare the native MIP model provided in Section 5, with and without the improvements of Subsection 5.4. We refer to these as imp and basic respectivelly. For solving the MIP models we use IBM ILOG Cplex (Cplex, 2009) version 12.9.0.

As we can see in Table 4, the implied constraints added in Section 5.4 slightly improve the efficiency of the approach allowing one more instance to be solved and improving the solving times of many other instances. Therefore, from now on we will use this improved model and refer to it simply as MIP.

7.4 Comparison

In this section we compare the best CP configuration, the best MaxSAT encoding and the best MIP formulation on the original instances as well as on crafted modifications from these. The crafted modifications add some properties that we think are worthy to take

6. https://github.com/marekpiotrow/UWrMaxSat.

Table 3: Solving times and best solutions found on original B2B instances with MaxSAT. Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

instance	basic		imp1		imp2		imp12		imp12$^+$	
forum-13	2.5	0	2.4	0	6.5	0	2.9	0	2.5	0
forum-13crafb	TO		80.6	6	248.1	6	37.5	6	23.3	6
forum-13crafc	17.8	1	2.7	1	43.9	1	22.4	1	2.3	1
forum-14	TO		14.6	2	TO		11.9	2	5.1	2
forumt-14	2.8	5	1.6	5	3.0	5	1.6	5	0.5	5
forumt-14crafc	9.4	5	1.8	5	4.1	5	1.6	5	0.9	5
forumt-14crafd	2.6	4	1.9	4	7.8	4	1.5	4	0.4	4
forumt-14crafe	TO		TO		92.4	5	1.5	5	0.6	5
tic-12	0.6	0	0.4	0	0.6	0	0.3	0	0.1	0
tic-12crafc	0.8	0	0.5	0	1.4	0	0.2	0	0.3	0
tic-13	1.1	0	1.6	0	2.5	0	1.5	0	0.4	0
tic-13crafb	1.3	0	1.4	0	1.2	0	1.4	0	0.4	0
tic-13crafc	TO		TO		200.7	4	7.6	4	5.8	4
tic-14crafa	2.5	0	1.7	0	2.9	0	1.8	0	1.0	0
tic-14crafc	51.3	0	3.5	0	5.9	0	1.5	0	0.8	0
tic-14crafd	4.0	0	1.8	0	2.7	0	1.5	0	0.8	0
ticf-13crafa	3.3	0	2.5	0	9.9	0	5.1	0	2.2	0
ticf-13crafb	822.2	3	90.2	3	213.8	3	31.3	3	22.3	3
ticf-13crafc	40.1	1	4.7	1	438.6	1	15.5	1	3.7	1
ticf-14crafa	TO		32.7	0	TO		12.4	0	7.6	0
#solved	15		18		18		20		20	

into account in the comparisons like, for instance, more constrained instances, unsatisfiable instances, higher optimums, etc.

7.4.1 Comparison on the Original Instances

In Table 5 we compare all best configurations and encodings of the considered approaches. As we can see, MaxSAT clearly dominates all other approaches since it is able to solve all the instances, sometimes orders of magnitude faster than the others. Notice that its worst reported solving time is of just 24 seconds (within a time-out of 7200 seconds). The next best approach is Chuffed with 18 solved instances, and then the native MIP model with 16. It is worth noticing that the CP model with Gecode and Cplex is not able to provide any upper bound for several instances while Chuffed and the native MIP model always report some solution. Therefore, Gecode and Cplex for CP are not used in the remaining experiments.

Next we extend the set of benchmarks by introducing some random modifications to each original B2B instance (forbidden time slots, fixed meetings, and meeting precedences). We also analyze how the value of the fairness parameter d affects the solvers' performance.

Table 4: Solving times and best solutions found on original B2B instances with MIP models, non-using improvements of Subsection 5.4 (basic) and using them (imp). Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

instance	basic		imp	
forum-13	3765.7	0	648.4	0
forum-13crafb	TO	22	TO	12
forum-13crafc	TO	6	TO	8
forum-14	TO	28	TO	21
forumt-14	173.1	5	44.9	5
forumt-14crafc	143.3	5	894.2	5
forumt-14crafd	1724.4	4	217.6	4
forumt-14crafe	TO	5	1311.8	5
tic-12	4.4	0	13.8	0
tic-12crafc	20.4	0	8.9	0
tic-13	3499.1	0	162.5	0
tic-13crafb	62.7	0	44.1	0
tic-13crafc	206.9	4	128.6	4
tic-14crafa	1443.6	0	326.3	0
tic-14crafc	302.8	0	628.1	0
tic-14crafd	872.3	0	194.7	0
ticf-13crafa	2012.5	0	1583.6	0
ticf-13crafb	TO	29	TO	17
ticf-13crafc	TO	14	TO	3
ticf-14crafa	TO	26	TO	22
#solved	13		14	

These modifications produce a set of 180 new B2B instances, that we add to the 20 original ones.

7.4.2 Forbidden Slots

The first modification introduced is on forbidden time slots. In particular, we modify the original B2B instances by varying the density α of forbidden time slots, with $0 \leq \alpha \leq 100$, where $\alpha = 0$ means that no participant has any forbidden time slot, and $\alpha = 100$ means that all participants have all time slots forbidden (thus the instance is trivially infeasible). We evaluate this modification with $\alpha = 0.3\%$ and $\alpha = 0.7\%$. Forbidden time slots are generated randomly. In order to generate these instances, we previously remove any existing forbidden time slot. We use such low values of α because we noticed that instances become unsatisfiable at very low densities. This is due to the existence of participants requesting as many meetings as available time slots. For them, there would be no feasible schedule if they had some forbidden time slot (this case never happens in the original instances). Experiments on these sets of instances are reported in Table 6.

Table 5: Solving times, and best solutions found, on original B2B problems with CP solvers. Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

instance	MaxSAT		CP						MIP	
			Chuffed		Gecode		Cplex			
forum-13	2.5	0	2891.2	0	TO	55	4174.0	0	648.4	0
forum-13crafb	23.3	6	1725.6	6	TO		TO		TO	12
forum-13crafc	2.3	1	TO	14	TO	56	TO	10	TO	8
forum-14	5.1	2	873.9	2	TO		TO		TO	21
forumt-14	0.5	5	5.6	5	TO	27	3201.5	5	44.9	5
forumt-14crafc	0.9	5	20.2	5	TO	24	3866.3	5	894.2	5
forumt-14crafd	0.4	4	20.9	4	TO		3726.6	4	217.6	4
forumt-14crafe	0.6	5	26.3	5	TO		TO	6	1311.8	5
tic-12	0.1	0	0.8	0	TO	10	258.0	0	13.8	0
tic-12crafc	0.3	0	1.2	0	TO	10	278.1	0	8.9	0
tic-13	0.4	0	16.9	0	TO	15	3595.7	0	162.5	0
tic-13crafb	0.4	0	2.0	0	TO	14	1842.4	0	44.1	0
tic-13crafc	5.8	4	518.2	4	TO		TO	4	128.6	4
tic-14crafa	1.0	0	22.4	0	TO		6955.3	0	326.3	0
tic-14crafc	0.8	0	24.1	0	TO		4986.6	0	628.1	0
tic-14crafd	0.8	0	6.2	0	TO		6375.9	0	194.7	0
ticf-13crafa	2.2	0	2473.4	0	TO	53	TO	5	1583.6	0
ticf-13crafb	22.3	3	2291.9	3	TO		TO		TO	17
ticf-13crafc	3.7	1	TO	13	TO	56	TO	19	TO	3
ticf-14crafa	7.6	0	1714.8	0	TO		TO		TO	22
#solved	20		18		0		11		14	

It can be observed that MaxSAT is the best solving approach, solving again all the instances. On some instances it is three orders of magnitude faster than the other approaches. However MIP is slightly faster for proving infeasibility, although it is the approach that solves the smallest number of benchmarks.

7.4.3 Fixed Meetings

In the second modification, we introduce randomly chosen fixed meetings. In particular, we vary the density β of fixed meetings, with $0 \leq \beta \leq 100$, where $\beta = 0$ means that no meeting is fixed and $\beta = 100$ corresponds to the instance in which all meetings have to be scheduled in a fixed time slot. In order to generate these instances, we select $\beta\%$ randomly chosen meetings, which are assigned to a randomly chosen time slot each. Additionally, we only allow fixed meetings in a time slot if none of the participants to the meeting has this time slot as forbidden or another fixed meeting in it. We evaluate this modification with $\beta = \{20, 40\}\%$. Again, bigger β results in a majority of unsatisfiable instances. Experiments on these sets of instances are reported in Table 7.

Table 6: Solving times, and best solutions found ('-' meaning unsatisfiability reported), on instances with a density of forbidden time slots $\alpha = 0.3\%$ and $\alpha = 0.7\%$. Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

instance	$\alpha = 0.3\%$						$\alpha = 0.7\%$					
	MaxSAT		Chuffed		MIP		MaxSAT		Chuffed		MIP	
forum-13	2.1	0	5841.6	0	1563.4	0	2.0	0	6161.0	0	2834.3	0
forum-13crafb	38.0	5	2819.4	5	TO	22	28.4	5	2714.2	5	TO	26
forum-13crafc	5.6	1	TO	14	TO	8	7.6	1	TO	13	TO	4
forum-14	6.8	0	2227.4	0	TO	34	0.7	-	8.0	-	0.1	-
forumt-14	1.0	0	24.0	0	244.2	0	0.3	-	2.5	-	0.1	-
forumt-14crafc	0.8	0	26.5	0	296.4	0	0.3	-	2.4	-	0.1	-
forumt-14crafd	0.3	-	2.8	-	0.1	-	0.3	-	2.5	-	0.1	-
forumt-14crafe	0.3	-	2.7	-	0.1	-	0.5	-	2.6	-	0.1	-
tic-12	0.2	-	0.7	-	0.1	-	0.1	-	1.1	-	0.1	-
tic-12crafc	0.3	-	0.7	-	0.1	-	0.7	-	0.7	-	0.1	-
tic-13	0.3	-	1.3	-	0.1	-	0.3	-	1.3	-	0.1	-
tic-13crafb	0.3	0	1.9	0	11.6	0	0.5	-	2.3	-	0.1	-
tic-13crafc	0.3	-	1.4	-	0.1	-	0.3	-	1.4	-	0.1	-
tic-14crafa	1.0	0	24.8	0	181.7	0	0.3	-	2.6	-	0.2	-
tic-14crafc	0.8	0	28.5	0	235.9	0	0.4	-	2.5	-	0.1	-
tic-14crafd	0.4	-	2.7	-	0.1	-	0.3	-	2.8	-	0.1	-
ticf-13crafa	2.3	0	5933.1	0	1042.1	0	2.0	0	5245.9	0	2548.6	0
ticf-13crafb	37.4	5	2799.8	5	TO	22	28.3	5	5385.6	5	TO	22
ticf-13crafc	5.2	1	TO	13	TO	7	7.4	1	TO	11	TO	4
ticf-14crafa	6.6	0	2776.4	0	TO	34	0.4	-	13.5	-	0.1	-
#solved	20		18		14		20		18		16	

The introduction of fixed meetings results in a significant increase of optimums, some of them reaching to 50 breaks. We have observed that some of the infeasibilities are due to the homogeneity constraint (notice that with a higher number of holes, fairness may become more difficult). This increase of optimums does not hurt at all the MaxSAT approach, which dominates again the other approaches and notably reduces the solving times compared to the original instances. This improvement in solving times also applies to Chuffed, which now is able to solve all the instances with both densities. Regarding MIP, it worsens its performance with $\beta = 20\%$ but it is able to solve all instances with $\beta = 40\%$. Notice that with $\beta = 40\%$ we already have more than half of instances unsatisfiable and all approaches are able to solve them easily.

7.4.4 Precedences

The third modification introduces random precedences among meetings. In this case, we modify the density γ of precedences among the meetings of all participants, with $0 \leq \gamma \leq 100$, where $\gamma = 0$ means that no meeting must precede another, whereas $\gamma = 100$

Table 7: Solving times, and best solutions found, on instances with a density of fixed meetings $\beta = 20\%$ and $\beta = 40\%$. Fairness is set to $d = 2$. TO stands for time-out (7200 seconds).

instance	$\beta = 20\%$						$\beta = 40\%$					
	MaxSAT		Chuffed		MIP		MaxSAT		Chuffed		MIP	
forum-13	1.8	20	90.9	20	1632.58	20	1.4	49	30.8	49	27.4	49
forum-13crafb	2.2	30	149.6	30	TO	33	0.4	-	0.1	-	0.1	-
forum-13crafc	1.8	20	327.3	20	TO	21	1.4	50	98.0	50	116.8	50
forum-14	4.7	20	725.9	20	TO	38	0.7	-	8.6	-	0.8	-
forumt-14	0.5	7	36.7	7	92.8	7	0.3	-	1.8	-	0.2	-
forumt-14crafc	0.5	7	30.3	7	307.2	7	0.4	-	1.8	-	0.1	-
forumt-14crafd	0.4	9	15.1	9	18.1	9	0.4	-	1.9	-	0.1	-
forumt-14crafe	0.5	9	19.1	9	40.1	9	0.3	-	2.1	-	0.1	-
tic-12	0.1	1	0.9	1	1.1	1	0.1	-	0.6	-	0.1	-
tic-12crafc	0.3	1	1.1	1	1.0	1	0.2	-	0.6	-	0.1	-
tic-13	0.4	2	1.9	2	47.2	2	0.5	13	13.3	13	6.0	13
tic-13crafb	0.2	1	3.2	1	23.5	1	2.0	16	2.2	16	1.4	16
tic-13crafc	1.0	6	19.7	6	13.2	6	15.7	16	1892.1	16	6.6	16
tic-14crafa	0.5	2	12.1	2	116.8	2	0.3	-	1.9	-	0.1	-
tic-14crafc	0.6	3	30.6	3	140.4	3	0.4	-	2.0	-	0.1	-
tic-14crafd	0.4	2	17.2	2	97.9	2	0.4		2.0	-	0.2	-
ticf-13crafa	2.1	20	93.0	20	3600.3	20	1.2	49	40.9	49	53.1	49
ticf-13crafb	2.5	27	208.9	27	TO	30	0.7	-	0.1	-	0.1	-
ticf-13crafc	2.0	20	532.2	20	TO	22	1.2	49	66.5	49	145.0	49
ticf-14crafa	4.9	21	791.5	21	TO	36	0.5	-	13.5	-	0.7	-
#solved	20		20		14		20		20		20	

corresponds to the case where, for each participant, all her meetings must be preceded by another (which is trivially infeasible). We evaluate this modification with $\gamma = \{15, 25\}\%$. Experiments on these sets of instances are reported in Table 8.

Once more MaxSAT is the best approach solving all the instances, but this modification makes some of them a bit harder than the original ones. Chuffed is the second best solving technique, but again MIP is the fastest at solving unsatisfiable instances.

7.4.5 Fairness

Finally, we analyze how the value of the fairness parameter d affects the results. To this purpose, we set the value of fairness to $d = \{0, 1, 3\}$. We recall that the value d represents the maximum difference in the number of breaks between any two participants. For instance, when $d = 0$, all participants are forced to have the *same* number of breaks. We recall that the experiments presented before correspond to the case where $d = 2$. Experiments on these sets of instances are reported in Tables 9, 10 and 11, respectively.

Table 8: Solving times, and best solutions found ('-' meaning unsatisfiability reported), on instances with a density of precedences among meetings $\gamma = 15\%$ and $\gamma = 25\%$. Fairness is set to $d = 2$.

instance	$\gamma = 15\%$						$\gamma = 25\%$					
	MaxSAT		Chuffed		MIP		MaxSAT		Chuffed		MIP	
forum-13	2.3	0	2053.0	0	4567.6	0	1.3	-	3.1	-	0.1	-
forum-13crafb	1.9	-	10.7	-	0.1	-	1.9	-	8.3	-	0.1	-
forum-13crafc	2.4	1	TO	12	TO	7	1.3	-	3.8	-	0.1	-
forum-14	21.9	2	2085.3	2	TO	34	2.5	-	7.8	-	0.1	-
forumt-14	0.4	5	37.8	5	504.2	5	1.0	5	176.1	5	2791.1	5
forumt-14crafc	0.9	5	42.7	5	569.1	5	1.5	5	127.8	5	1195.2	5
forumt-14crafd	0.4	4	46.4	4	TO	6	1.4	4	63.4	4	1068.7	4
forumt-14crafe	1.0	5	49.7	5	TO	5	0.9	5	111.0	5	5440.1	5
tic-12	0.2	0	1.6	0	12.6	0	0.2	0	8.6	0	218.3	0
tic-12crafc	0.3	0	1.5	0	9.0	0	0.6	0	18.9	0	267.7	0
tic-13	0.4	0	27.9	0	243.0	0	0.8	0	65.3	0	4698.0	0
tic-13crafb	0.7	0	27.5	0	846.0	0	0.6	0	57.6	0	565.8	0
tic-13crafc	6.8	4	485.6	4	195.8	4	6.7	4	873.5	4	3001.2	4
tic-14crafa	1.1	0	62.0	0	661.8	0	1.4	0	153.8	0	TO	2
tic-14crafc	0.8	0	44.0	0	1265.0	0	1.3	0	133.4	0	TO	2
tic-14crafd	0.8	0	52.7	0	787.7	0	1.5	0	107.4	0	TO	5
ticf-13crafa	2.8	0	4185.2	0	4795.9	0	1.8	-	3.3	-	0.1	-
ticf-13crafb	2.1	-	6.6	-	0.1	-	2.1	-	5.0	-	0.1	-
ticf-13crafc	3.9	1	TO	12	TO	7	1.7	-	3.3	-	0.1	-
ticf-14crafa	170.5	0	7013.5	0	TO	45	2.7	-	11.8	-	0.1	-
#solved	20		18		14		20		20		17	

First of all we can observe that with $d = 0$ there are a lot of unsatisfiable instances. Notice that there are instances with participants that cannot have any break in their schedules (because they have just one meeting or they have as many meeting as time periods). Therefore, all the problems with an optimum bigger than 0 in their corresponding original instance will become unsatisfiable by setting the fairness to $d = 0$. Solving times are smaller for all the approaches. In fact, in this setting MIP is the second best, being able to close all the instances. For $d = 1$ and $d = 3$, no unsatisfiable instance appears and solving times are quite similar than with $d = 2$. MaxSAT is again the best solving approach.

8. Conclusions

We have precisely formulated the Business-to-Business Meeting Scheduling Optimization problem (B2BSOP) and presented a comparative study of different model-and-solve exact approaches to this problem. In particular, we have evaluated CP, MaxSAT and MIP formulations, and considered distinct CP solving technologies. The considered approaches are

Table 9: Solving times, and best solutions found ('-' meaning unsatisfiability reported), on instances with fairness $d = 0$.

instance	MaxSAT		Chuffed		MIP	
forum-13	2.1	0	1190.4	0	38.9	0
forum-13crafb	1.4	-	7.2	-	0.4	-
forum-13crafc	1.6	-	TO		106.1	-
forum-14	1.6	-	10.8	-	0.4	-
forumt-14	0.2	-	5.5	-	0.1	-
forumt-14crafc	0.2	-	4.6	-	0.1	-
forumt-14crafd	0.2	-	5.0	-	0.1	-
forumt-14crafe	0.2	-	4.5	-	0.2	-
tic-12	0.2	0	2.3	0	0.2	0
tic-12crafc	0.2	0	2.3	0	0.1	0
tic-13	0.3	0	6.1	0	2.8	0
tic-13crafb	0.3	0	5.8	0	0.2	0
tic-13crafc	0.2	-	4.2	-	1.1	-
tic-14crafa	0.6	0	5.6	0	0.6	0
tic-14crafc	0.6	0	4.5	0	2.6	0
tic-14crafd	0.3	0	5.5	0	0.8	0
ticf-13crafa	2.3	0	3595.5	0	38.4	0
ticf-13crafb	1.5	-	8.4	-	0.6	-
ticf-13crafc	2.2	-	TO		914.8	-
ticf-14crafa	6.3	0	67.2	0	3736.4	0
#solved	20		18		20	

refinements and improvements of the best approaches as reported by Pesant et al. (2015) and Bofill et al. (2015). In particular:

- We consider some extensions of the problem with additional constraints.
- We provide a complete MiniZinc model, and details on how to build the three variants tested in the experiments.
- We provide a much more compact automaton to identify participants' schedules breaks for optimization.
- We compare the performance of the four MiniZinc models with three distinct solving technologies: CP (Gecode), Lazy Clause Generation (Chuffed) and Linear Integer Programming (Cplex).
- We provide further details of the MaxSAT model and extend it to deal with the new constraints.
- We provide two more refinements of the encoding that improve even more the MaxSAT approach performance.

Table 10: Solving times, and best solutions found, on instances with fairness $d = 1$.

instance	MaxSAT		Chuffed		MIP	
forum-13	1.8	0	3604.0	0	616.0	0
forum-13crafb	43.4	6	3071.2	6	TO	21
forum-13crafc	2.4	1	TO	13	TO	6
forum-14	6.2	2	1600.0	2	TO	45
forumt-14	0.4	5	5.8	5	153.8	5
forumt-14crafc	0.5	5	20.8	5	216.3	5
forumt-14crafd	0.5	4	17.3	4	TO	5
forumt-14crafe	0.6	5	21.3	5	2016.3	5
tic-12	0.1	0	0.9	0	10.8	0
tic-12crafc	0.1	0	1.1	0	10.4	0
tic-13	0.3	0	12.9	0	13.5	0
tic-13crafb	0.2	0	2.4	0	31.6	0
tic-13crafc	6.1	4	516.2	4	98.1	4
tic-14crafa	0.7	0	21.5	0	37.9	0
tic-14crafc	0.7	0	16.0	0	301.2	0
tic-14crafd	0.6	0	5.1	0	177.5	0
ticf-13crafa	2.3	0	1629.6	0	1381.6	0
ticf-13crafb	28.9	3	2475.1	23	TO	26
ticf-13crafc	4.4	1	TO	11	TO	3
ticf-14crafa	6.6	0	1432.12	0	TO	50
#solved	20		18		13	

- We improve the MIP model by Pesant et al. (2015), and adapt it to deal with the new constraints.

The considered dataset consists of industrial B2B instances, with some variants including constraints like precedences between meetings and forbidden time slots. We contribute 180 new instances. Some of these variants increase the number of breaks of participants but do not make optimization harder. Experimental results show that the MaxSAT approach is state-of-the-art for this problem among the considered technologies. Interestingly, we have observed that the use of some implied constraints in the MaxSAT encodings improves their performance, allowing to solve all the instances of the extended benchmark set. Among CP solving technologies, Lazy Clause Generation dominates all other approaches, and using the smallest automaton on the reified schedules turns to be the best configuration for all solvers. Gecode is not competitive in the considered datasets, not being able to solve any instance within the given time limit. Using Cplex to solve the CP formulation provides a worse performance than using the native MIP model.

As further work we consider to investigate other variants of the B2BSOP, such as considering meetings with more than two participants, or allowing a participant to have more than one meeting at a time (assuming, for instance, that a company is sending two representatives to the brokerage event). Another interesting property of the schedules is to minimize the number of location changes that participants have to do. This was prelimi-

Table 11: Solving times, and best solutions found, on instances with fairness $d = 3$.

instance	MaxSAT		Chuffed		MIP	
forum-13	2.3	0	1419.6	0	1026.9	0
forum-13crafb	11.9	6	1401.6	6	TO	17
forum-13crafc	2.7	1	TO	12	TO	3
forum-14	7.2	2	1064.0	2	TO	53
forumt-14	0.4	5	4.8	5	213.0	5
forumt-14crafc	0.8	5	24.3	5	246.1	5
forumt-14crafd	0.5	4	21.2	4	87.5	4
forumt-14crafe	0.5	5	38.7	5	814.6	5
tic-12	0.2	0	0.8	0	18.4	0
tic-12crafc	0.2	0	1.7	0	6.2	0
tic-13	0.6	0	21.9	0	189.3	0
tic-13crafb	0.3	0	2.1	0	80.0	0
tic-13crafc	3.9	4	499.7	4	87.2	4
tic-14crafa	0.7	0	31.5	0	227.4	0
tic-14crafc	1.0	0	37.9	0	143.1	0
tic-14crafd	0.6	0	5.9	0	195.5	0
ticf-13crafa	2.6	0	5917.4	0	2964.9	0
ticf-13crafb	27.0	3	3328.2	3	TO	11
ticf-13crafc	4.4	1	TO	10	TO	14
ticf-14crafa	7.5	0	4211.5	0	TO	30
#solved	20		18		14	

narily studied by Bofill et al. (2014), but we believe that a more integrated approach should be considered.

Acknowledgments

Work partially supported by grants PGC2018-101216-B-I00 and RTI2018-095609-B-I00 (MICINN/FEDER, UE), fellowships FJCI-2017-32420, IJC2019-040489-I (MICINN/Juan de la Cierva), *Ayudas para Contratos Predoctorales 2016* (grant number BES-2016-076867, funded by MINECO and co-funded by FSE), AMU Institut Archiméde and NSERC Discovery Grant 218028/2017.

We thank the anonymous referees for their useful comments that helped us to improve the paper. We were able to obtain a better performance of the Minizinc with Chuffed solver approach thanks to several suggestions from them.

Appendix. MiniZinc Models

In this appendix we provide MiniZinc models for the CP approach. Namely, we fully provide and comment the MiniZinc model using automaton $\mathcal{A}_1$ without meeting/no-meeting reification. Finally we also provide the changes needed in order to use this reification and automaton $\mathcal{A}_0$.

MiniZinc Model without Reification and with Automaton $\mathcal{A}_1$

MiniZinc models start with global constraints inclusions and follow with the parameters. Parameters get values from each particular instance defined in corresponding `dzn` files.

We remark that, although the meetings in the parameters `requested`, `fixed` and `precedences` are indexed from `1` to `nMeetings`, and `precedences` range over `set of 1..nMeetings`, parameter `meetingsxBusiness` and variable `s` (in Listing 2) shift by one the meeting references and range from `1` to `nMeetings+1`. We do this because the MiniZinc global constraint `cost_regular` does not allow `0`s in the sequence to recognize. Recall that automata defined in Section 3.2 use `0` as no-meeting. Then, thanks to this shifting, we can use now `1` as "no-meeting" instead of `0`.

Listing 1: Parameters

```
include "globals.mzn";

% PARAMETERS
int: nBusiness;    % number of participants
int: nMeetings;    % number of meetings
int: nTables;      % max number of simultaneous meetings
int: nTotalSlots;   % number of timeslots
int: nMorningSlots; % number of morning slots
int: diference;     % diference for fairness
array[1..nMeetings,1..3] of int: requested; % two participants
  and  preferred session (1 morning, 2 afternoon, 3 no
  preference) of each meeting
array[1..nBusiness] of 0..nTotalSlots: nMeetingsBusiness; %
  number of meetings of each participant
array[1..nBusiness] of set of 1..nMeetings+1: meetingsxBusiness;
   % set of (shifted + 1) meetings of each business, (where 1 =
  no meeting)
array[1..nMeetings] of 0..nTotalSlots: fixed; % fixed timeslot
  for each meeting (0 = no fixed)
array[1..nBusiness] of set of 0..nTotalSlots: forbidden;  % set
  of forbidden timeslots per participant
array[1..nMeetings] of set of 1..nMeetings: precedences;  % set
  of meetings preceding each meeting
```

Then, variables follow. For each variable we define its domain, which typically depends on parameters.

Listing 2: Variables

```
% VARIABLES
array[1..nBusiness,1..nTotalSlots] of var 1..nMeetings+1: s;  %
  main variable stating for each business and  timeslot which
  (shifted +1) meeting is scheduled (1 = no meeting)
array[1..nTotalSlots,1..nMeetings+1] of var 0..nBusiness:
  countMeetings; % auxiliar varible to count the number of
  occurrences of (shifted +1) meetings in the schedules (1 = no
  meeting), its domain is bounded in forthcoming constraint with
  [nBusiness-2*nTables..nBusiness,{0,2}....{0,2}]
array[1..nBusiness] of var 0..nTotalSlots-1: b; % auxiliary cost
  variable for each business
array[1..nMeetings] of var 1..nTotalSlots: timeOfMeeting; %
  timeslot in which each meeting is held
var 0..nMeetings: mini; % auxiliary variable with the minimum
  number of breaks among all participants used to enforce fairness
var 0..nMeetings: cost; % total cost variable to minimize
```

Then constraints follow. We first introduce feasibility related constraints and then optimization ones.

As explained in Section 3, feasibility is basically dealt with by using the global constraint `global_cardinality`. Recall that `global_cardinality(seq, values, occur)`, is enforcing that each `values[i]` occurs exactly `occur[i]` times in `seq` . Most of these parameters are created with list comprehensions.

Listing 3: Feasibility Constraints

```
% CONSTRAINTS
% Feasibility Constraints
% Refine the domain of variable s according to meetings of each
  business
constraint
  forall (b in 1..nBusiness)(
    forall (t in 1..nTotalSlots)( s[b,t] in meetingsxBusiness[b]
      )
  );

% Bound the possible occurrences of meetings in the schedule
  using auxiliary variable countMeetings to be used in the
  forthcoming global_cardinality constraint. No-meeting (1 value)
  will occur at least nBusiness-2*nTables times, and all other
  meetings can occur 0 or 2 times.
constraint
  forall (t in 1..nTotalSlots)(
    forall (m in 1..nMeetings+1)(
      if m==1 then
```

```
        countMeetings[t,m] in (nBusiness-2*nTables)..nBusiness
      else
        countMeetings[t,m] in {0,2}
      endif
    )
  );
constraint
  forall (t in 1..nTotalSlots)(
    nBusiness-sum(p in 2..nMeetings+1) (countMeetings[t,p]) ==
     countMeetings[t,1]
  );

% Constraint ensuring that each participant has all her meetings
  scheduled and no more (counting the number of occurrences of
  1s).
constraint
  forall (p in 1..nBusiness)(
    global_cardinality(
      [s[p,t] | t in 1..nTotalSlots],
      [i | i in 1..nMeetings+1],
      [nTotalSlots-nMeetingsBusiness[p]]++
        [if requested[m-1,1]==p \/ requested[m-1,2]==p then 1
         else 0 endif | m in 2..nMeetings+1])
  );

% Constraint ensuring that participants of a meeting in a same
  timeslot are 0 or two (meeting 1 is a special case that depends
  on the number of participants, meetings and total timeslots)
constraint
  forall (t in 1..nTotalSlots)(
    global_cardinality(
      [s[p,t]|p in 1..nBusiness],
      [i|i in 1..nMeetings+1],
      [countMeetings[t,m]|m in 1..nMeetings+1])
  );
```

Again, as explained in Section 3, optimization is basically dealt with by using the `cost_regular` global constraint. This constraint is only applied to businesses `p` that may have schedules with holes, i.e., businesses with more than one meeting and less than `nTotalSlots` meetings, otherwise `b[p]==0`. We recall that the MiniZinc global constraint `cost_regular(seq,nest,alpha,trans,ini,final,trans_cost,cost)` ensures the sequence `seq` to be a word made of numbers from 1 to `alpha` belonging to the regular language defined by the automaton with `nest` states, initial state `ini`, set of final states `final` and transitions `trans`. These transitions are described as a 2D array stating, for each state and possible symbol of the alphabet (in the considered code, `1..nMeetings+1`), which is the destination state (state `0` is the *error state*). In addition, `cost` is summing up

the cost of each transition used to accept the sequence. The cost of the transitions is defined in parameter `trans_cost` as an array of costs per state and symbol. In the provided code we are defining automaton $\mathcal{A}_1$ and we look for breaks over the sequences `[s[p,t]|t in 1..nTotalSlots]` of meetings scheduled in all time slots of each participant `p`. Once more, most of these parameters are created with list comprehensions. Notice however the use of `array2d` to transform lists to 2D arrays.

Listing 4: Optimization

```
% Optimization Constraints
% cost_regular with A_1 Automaton and considering variable s to
  identify breaks
% cost will sum the costs of all participants
constraint
  forall (p in 1..nBusiness)(
    if nMeetingsBusiness[p] <= 1 \/ nMeetingsBusiness[p] ==
      nTotalSlots then
      b[p]==0
    else
      cost_regular(
        [s[p,t]|t in 1..nTotalSlots],
        2*nMeetingsBusiness[p],
        nMeetings+1,
        array2d(1..2*nMeetingsBusiness[p],1..nMeetings+1,
                [1]++[if i in meetingsxBusiness[p] then 2
                      else 0 endif|i in 2..nMeetings+1]++
                [if e mod 2==0 then
                    if a==1 then
                        e+1
                    else  if a in meetingsxBusiness[p] then
                                e+2
                          else
                                0
                          endif
                    endif
                 else if a==1 then
                          e
                      else  if a in meetingsxBusiness[p] then
                                  e+1
                            else
                                  0
                            endif
                      endif
                endif|e in 2..nMeetingsBusiness[p]*2-1,a in
                  1..nMeetings+1]++
```

```
                [2*nMeetingsBusiness[p]]++[0|i in
                 2..nMeetings+1] ),
        1,
        {2*nMeetingsBusiness[p]},
        array2d(1..2*nMeetingsBusiness[p],1..nMeetings+1,
                [0|i in 1..nMeetings+1]++
                [if e mod 2==0 /\ a==1 then 1 else 0 endif|e in
                 2..nMeetingsBusiness[p]*2-1,a in
                 1..nMeetings+1]++
                [0|i in 1..nMeetings+1]),
        b[p])
    endif
  );

constraint
   cost == sum(p in 1..nBusiness) (b[p]);
```

Finally, the remaining constraints dealing with fixed sessions, fixed meetings, forbidden time slots, precedences and fairness.

Listing 5: Additional constraints.

```
% Additional Constraints
% fixed session
constraint
  forall (m in 1..nMeetings)(
    if requested[m,3]==1 then
      forall (j in nMorningSlots+1..nTotalSlots)
          (s[requested[m,1],j] != m+1)
    else
      if requested[m,3]==2 then
        forall (j in 1..nMorningSlots)
            (s[requested[m,1],j] != m+1)
      else
        true
      endif
    endif
  );

% fixed meeting
constraint
  forall (m in 1..nMeetings)(
    if fixed[m]!=0 then
      (s[requested[m,1],fixed[m]] == m+1) /\
      (s[requested[m,2],fixed[m]] == m+1)
    else
      true
```

```
    endif
  );

% forbidden timeslots
constraint
  forall (p in 1..nBusiness)(
    forall (j in forbidden[p])(
      if j!=0 then s[p,j]==1 else true endif
    )
  );

% precedences
constraint forall(p in 1..nBusiness, m in meetingsxBusiness[p]
 where m!=1)
       (s[p,timeOfMeeting[m-1]] = m);

constraint forall(m in 1..nMeetings, j in precedences[m])
       (timeOfMeeting[j] < timeOfMeeting[m]);

% Fairness
constraint
  if diference>=0 then
     minimum(mini,b) /\
     forall (p in 1..nBusiness)(b[p]- mini <= diference)
   else
    true
  endif;
```

We also show the search strategy used with Chuffed and the objective function.

Listing 6: Search strategy for Chuffed.

```
solve :: seq_search([int_search([cost], input_order,
 indomain_min),int_search(s, input_order, indomain_min)])
 minimize cost; % search strategy and objective for Chuffed
```

Using Reification and Automaton $\mathcal{A}_0$

Taking the previous Minizinc model, we just need to replace the automaton $\mathcal{A}_1$ of Listing 4 by the following one for $\mathcal{A}_0$:

Listing 7: Automaton $\mathcal{A}_0$ with Meeting/No-meeting Reification.

```
constraint
   forall (p in 1..nBusiness)(
     if nMeetingsBusiness[p] <= 1 \/ nMeetingsBusiness[p] ==
      nTotalSlots then
       b[p]==0
```

```
     else
      cost_regular([ bool2int(s[p,t] > 1) + 1 | t in
       1..nTotalSlots ], 3, 2,
       [|1,2|3,2|3,2|],
       1, 1..3,
       [|0,0|0,0|0,1|],
       b[p])
    endif
);
```

Notice that here, instead of directly working with the sequence of multivalued variables `[s[p,t]|t in 1..nTotalSlots]`, we deal with an ad-hoc sequence of Boolean values `[bool2int(s[p,t]>1)+1|t in 1..nTotalSlots]` which indicate whether meeting `p` has a meeting or not at a certain time slot.

Global Constraints with Specialized Propagators for Chuffed

Chuffed solver has specialized propagators for the `global_cardinality_low_up` constraint. In order to take advantage of them, we implement constraint (4) as follows, in replacement of the first `global_cardinality` in Listing 3:

Listing 8: Minizinc code for Constraint (4)

```
constraint
   forall (p in 1..nBusiness)(
     global_cardinality_low_up([s[p,t]|t in 1..nTotalSlots],
        [i|i in 1..nMeetings+1],
        [nTotalSlots-nMeetingsBusiness[p]]++
           [if requested[m-1,1]==p \/ requested[m-1,2]==p then 1
            else 0 endif|m in 2..nMeetings+1],
        [nTotalSlots-nMeetingsBusiness[p]]++
           [if requested[m-1,1]==p \/ requested[m-1,2]==p then 1
            else 0 endif|m in 2..nMeetings+1])
   );
```

The `global_cardinality_low_up` constraint alone does not allow to model Constraints (5) but, in order to improve propagation, we can add the following as a redundant constraint in addition to the second `global_cardinality` in Listing 3:

Listing 9: Redundant Minizinc code for Constraint (5)

```
constraint
    forall (t in 1..nTotalSlots)(
      global_cardinality_low_up([s[p,t]|p in 1..nBusiness],
       [i|i in 1..nMeetings+1],
       [nBusiness-2*nTables]++[0|m in 1..nMeetings],
       [nBusiness]++[2|m in 1..nMeetings])
    );
```

References

Abío, I., Nieuwenhuis, R., Oliveras, A., & Rodríguez-Carbonell, E. (2013). A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In *19th International Conference on Principles and Practice of Constraint Programming, CP 2013*, Vol. 8124 of *LNCS*, pp. 80–96. Springer.

Bofill, M., Espasa, J., Garcia, M., Palahí, M., Suy, J., & Villaret, M. (2014). Scheduling B2B Meetings. In *20th International Conference on Principles and Practice of Constraint Programming, CP 2014*, Vol. 8656 of *LNCS*, pp. 781–796. Springer.

Bofill, M., Garcia, M., Suy, J., & Villaret, M. (2015). MaxSAT-based scheduling of B2B meetings. In *Proc. of CPAIOR 2015*, pp. 65–73.

Chu, G. (2011). *Improving combinatorial optimization*. Ph.D. thesis, The University of Melbourne. http://hdl.handle.net/11343/36679.

Cplex, I. I. (2009). *V12.1: Users Manual for CPLEX*.

Demassey, S., Pesant, G., & Rousseau, L. (2006). A Cost-Regular Based Hybrid Column Generation Approach. *Constraints*, *11*(4), 315–333.

Even, S., Itai, A., & Shamir, A. (1975). On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 16th Annual Symposium*, pp. 184–193. IEEE.

Gebser, M., Glase, T., Sabuncu, O., & Schaub, T. (2013). Matchmaking with Answer Set Programming. In *12th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, Vol. 8148 of *LNCS*, pp. 342–347. Springer.

Klieber, W., & Kwon, G. (2007). Efficient CNF encoding for selecting 1 from N objects. In *Fourth Workshop on Constraints in Formal Verification, CFV*.

Li, C. M., & Manyà, F. (2009). *Handbook of Satisfiability*, chap. MaxSAT, Hard and Soft Constraints, pp. 613–631. IOS Press.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: Towards a Standard CP Modelling Language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, Vol. 4741 of *LNCS*, pp. 529–543. Springer.

Pesant, G., Rix, G., & Rousseau, L. (2015). A comparative study of MIP and CP formulations for the B2B scheduling optimization problem. In *Proc. of CPAIOR 2015*, pp. 306–321.

Piotrów, M. (2019). UWrMaxSat-a new MiniSat+-based Solver in MaxSAT Evaluation 2019. MaxSAT Evaluation 2019.

Régin, J.-C. (1996). Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of the Thirteenth National/Eighth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI-98/IAAI-98*, Vol. 1, pp. 209–215.

Schulte, C., Tack, G., & Lagerkvist, M. Z. (2019). Modeling and Programming with Gecode. www.gecode.org.

Sinz, C. (2005). Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *11th International Conference on Principles and Practice of Constraint Programming, CP 2005*, Vol. 3709 of *LNCS*, pp. 827–831. Springer.