A Study on Implied Constraints in a MaxSAT Approach to B2B Problems *

Miquel Bofill[1], Marc Garcia[1], Jesús Giráldez-Cru[2], and Mateu Villaret[1]

[1] Departament d'Informàtica, Matemàtica Aplicada i Estadística,
Universitat de Girona, Spain
{mbofill,mgarciao,villaret}@imae.udg.edu
[2] Artificial Intelligence Research Institute (IIIA-CSIC), Spain
jgiraldez@iiia.csic.es

Abstract

The B2B Scheduling Optimization Problem (B2BSOP) consists in finding the schedule of a set of meetings between persons minimizing the waiting time periods between meetings of the participants. Recent results have shown that a SAT-based approach is the-state-of-the-art. One of the interesting features of such approach is the use of implied constraints. In this work, we provide an experimental setting to study the effectiveness of using those implied constraints. Since there exists a reduced number of real-world B2B instances, we propose a random B2B instances generator, which reproduces certain features of these known to date real-world instances. We show the usefulness of some implied constraints depending on the characteristics of the problem, and the benefits of combining them. Finally, we give some insights on the exploitation of these implied constraints by the solver.

1 Introduction

Business-to-business (B2B) events consist in meetings between people with similar interests. They are typically held in business networking events. The B2B scheduling problem is the problem of finding a feasible schedule for a set of requested meetings between participants, subject to participant availability and accommodation capacity. To the best of our knowledge, there are few works dealing with this problem. In [6], authors propose a system for computing such schedules in several fairs.

The B2B scheduling optimization problem (B2BSOP) considered in this work is introduced in [4]. It consists in finding a feasible B2B schedule that minimizes the number of breaks in participants' schedule (or idle time periods), cumulatively over all participants. Side constraints can be added to ensure fairness among participants, e.g., restricting the maximum difference in the number of breaks among all participants' schedules. A further refinement could consist in (additionally or alternatively) minimizing the duration of the breaks.

The B2BSOP has been successfully addressed on real-world instances by means of relatively straightforward CP and Pseudo-Boolean formulations [4], and further investigated in [10] with specialized CP and MIP formulations and in [5] with partial MaxSAT specialized encodings. In a partial MaxSAT formula [8], some clauses are marked as hard whilst 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 the encoding from [5], which we will use in this paper, the falsification of a soft clause will represent the existence of an idle time period

*This work is partially funded by the MINECO/FEDER UE project LoCoS (TIN2015-66293-R), the UdG project MPCUdG2016/055, the CSIC project 201450E045, and MINECO/FEDER UE project RASO (TIN2015-71799-C2-1-P).

for some participant. This MaxSAT encoding clearly dominates the other formulations, and the use of implied constraints appears to be beneficial.

There already exist some works in the literature showing the benefits of using implied constraints or making redundant encodings in SAT, see for instance [7, 2, 3]. However, the lack of varied real-world B2B instances makes difficult to hold this claim for the problem at hand. In this paper, we face this problem by means of random generators. The development of random problems generators sharing the majority of real-world instance features was already stated in [11] as one of the most important challenges for the next few years. The main goal of such generators is two-sided. On one hand, these generators allow us to augment the set of problems since the random generated instances share the main characteristics of the known problems. On the other hand, they allow us to create instances of any desired size. This is interesting for testing and debugging purposes, since we can create families of instances easier to solve, and hence any technique evaluation can be performed faster.

In this work, we propose a generator of random B2B instances. It is parameterized in several ways, in order to reproduce certain characteristics of B2B real-world problems in a customized fashion. This allows us to get some insights on the strengths and weaknesses of those implied constraints based on the characteristics of the problem. Namely, we show the benefits of using implied constraints with respect to the density and the shape of the problem. By density we mean the ratio between the number of meetings and the accommodation capacity. By shape we mean the configuration of the accommodation capacity (i.e., ratio between time slots and locations). Finally, we illustrate how the use of implied constraints modifies the branching variable selection of the solver, and we conjecture its possible relation to the successful solver performance.

2 Background

Here we precisely reproduce the definition of the problem at hand used in [4].

Definition 1 (B2BSOP-h)**.** *Given a set of participants $\mathcal{P}$, a list of time slots $\mathcal{T}$, a set of available locations $\mathcal{L}$ and a set of meetings $\mathcal{M} \subset \mathcal{P} \times \mathcal{P}$ between pairs of participants to be scheduled, where participants may have forbidden meeting time slots and meetings may have morning or afternoon celebration requirements, the* B2B Scheduling Optimization Problem with homogeneity h *consists in finding a total mapping from $\mathcal{M}$ to $\mathcal{T} \times \mathcal{L}$, minimizing the total number of idle time periods cumulatively over all participants, and such that:*

- *Each participant has at most one meeting scheduled in each time slot.*
- *No meeting is scheduled in a forbidden time slot for any of its participants.*
- *At most one meeting is scheduled in a given time slot and location.*
- *Each meeting having a morning or afternoon celebration requirement is scheduled in a time slot of the appropriate interval.*
- *The difference between the number of idle time periods among all participants is at most h, where by an* idle time period *we refer to a group of consecutive idle time slots between two consecutive meetings involving the same participant.*

In the MaxSAT encoding presented in [5], among others, there are $|\mathcal{M}| \cdot |\mathcal{T}|$ Boolean variables $schedule_{i,j}$ representing if meeting i is held in time slot j, and $|\mathcal{P}| \cdot |\mathcal{T}|$ Boolean variables $usedSlot_{p,j}$ representing if participant p has a meeting scheduled in time slot j.

For readability, we also reproduce a simplified[1] version of this MaxSAT encoding.

Parameters. Each instance is defined by the following parameters:

- $nMeetings$: number of meetings.
- $nTimeSlots$: number of available time slots.
- $nTables$: number of available locations.
- $nParticipants$: number of participants.
- $meetings$, function from $\{1, \ldots, nParticipants\}$ to $2^{\{1,\ldots,nMeetings\}}$: set of meetings involving each participant.
- $forbidden$, function from $\{1, \ldots, nParticipants\}$ to $2^{\{1,\ldots,nTimeSlots\}}$: set of forbidden time slots for each participant.

Variables. We define the following propositional variables:

- $schedule_{i,j}$: meeting i is held in time slot j.
- $usedSlot_{p,j}$: participant p has a meeting scheduled in time slot j.
- $fromSlot_{p,j}$: participant p has a meeting scheduled at, or before, time slot j.
- $endHole_{p,j}$: participant p has an idle time period finishing at time slot j.

We also use some auxiliary variables that will be introduced when needed.

In what follows, we use several Boolean global constraints with the following meaning. $exactly(k, S)$ states that exactly k variables in S are set to true, $atMost(k, S)$ states that at most k variables in S are set to true, and finally, $sortingNetwork(S, sl)$ states that sl is the sorted list of Boolean variables of S.

Constraints. All constraints are hard if not explicitly defined as soft constraints. To help readability we define $\mathcal{M} = \{1, \ldots, nMeetings\}$, $\mathcal{T} = \{1, \ldots, nTimeSlots\}$, $\mathcal{P} = \{1, \ldots, nParticipants\}$.

At most one meeting involving the same participant is scheduled in each time slot.

$$atMost(1, \{schedule_{i,j} \mid i \in meetings(p)\}) \quad \forall p \in \mathcal{P}, j \in \mathcal{T}$$

No meeting is scheduled in a forbidden time slot for any of its participants.

$$\bigwedge_{i \in meetings(p),\, j \in forbidden(p)} \neg schedule_{i,j} \qquad \forall p \in \mathcal{P}$$

Each meeting is scheduled in exactly one time slot.

$$exactly(1, \{schedule_{i,j} \mid j \in \mathcal{T}\}) \qquad \forall i \in \mathcal{M}$$

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

$$atMost(nTables, \{schedule_{i,j} \mid i \in \mathcal{M}\}) \qquad \forall j \in \mathcal{T}$$

In order to be able to minimize the number of idle time periods we introduce channeling constraints between the variables *schedule*, *usedSlot* and *fromSlot*.

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

$$schedule_{i,j} \rightarrow (usedSlot_{p_i^1,j} \wedge usedSlot_{p_i^2,j}) \quad \forall i \in \mathcal{M}, j \in \mathcal{T}$$

[1] For instance, we do not show the encodings of morning/afternoon requirements. The complete and detailed encoding can be found in [5].

where p_i^1 and p_i^2 are the participants of meeting i.

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

$$usedSlot_{p,j} \rightarrow \bigvee_{i \in meetings(p)} schedule_{i,j} \qquad \forall p \in \mathcal{P}, j \in \mathcal{T}$$

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

$$\neg usedSlot_{p,1} \rightarrow \neg fromSlot_{p,1} \qquad \forall p \in \mathcal{P}$$

$$(\neg fromSlot_{p,j-1} \wedge \neg usedSlot_{p,j}) \rightarrow \neg fromSlot_{p,j} \qquad \forall p \in \mathcal{P}, j \in T \setminus \{1\}$$

$$usedSlot_{p,j} \rightarrow fromSlot_{p,j} \qquad \forall p \in \mathcal{P}, j \in \mathcal{T}$$

$$fromSlot_{p,j-1} \rightarrow fromSlot_{p,j} \qquad \forall p \in \mathcal{P}, j \in \mathcal{T} \setminus \{1\}$$

Optimization. Minimization of the number of idle time periods is achieved by means of soft constraints.

An idle time period can be identified when a participant has no meeting in a certain time slot, has a meeting in the next time slot, and has had a meeting before. We reify this situation with the *endHole* variables.

$endHole_{p,j}$ is true if and only if participant p has an idle time period finishing at time slot j.

$$endHole_{p,j} \leftrightarrow \left(\neg usedSlot_{p,j} \wedge fromSlot_{p,j} \wedge usedSlot_{p,j+1}\right) \qquad \forall p \in \mathcal{P}, j \in \mathcal{T} \setminus \{nTimeSlots\}$$

$sortedHole_{p,1}, \ldots, sortedHole_{p,\lfloor (nTimeSlots-1)/2 \rfloor}$ are the unary representation of the number of idle time periods of each participant p.[2]

$$sortingNetwork([endHole_{p,j} \mid j \in \mathcal{T}], [sortedHole_{p,j} \mid j \in \mathcal{T}]) \qquad \forall p \in \mathcal{P}$$

[Soft Constraint] *A participant does not have idle time periods.*

$$\neg sortedHole_{p,j} \qquad \forall p \in \mathcal{P}, j \in 1..\lfloor (nTimeSlots - 1)/2 \rfloor$$

Homogeneity. We find the maximum and minimum number of idle time periods among all participants, and enforce homogeneity by bounding their difference.

(An approximation to) The unary representation of the maximum and minimum number of idle time periods among all participants are respectively $max_1, \ldots, max_{\lfloor (nTimeSlots-1)/2 \rfloor}$ and $min_1, \ldots, min_{\lfloor (nTimeSlots-1)/2 \rfloor}$.

$$sortedHole_{p,j} \rightarrow max_j \ \forall p \in \mathcal{P}, j \in 1..\lfloor (nTimeSlots - 1)/2 \rfloor$$
$$\neg sortedHole_{p,j} \rightarrow \neg min_j \ \forall p \in \mathcal{P}, j \in 1..\lfloor (nTimeSlots - 1)/2 \rfloor$$

[2] Notice that the number of possible idle time periods of each participant is strictly smaller than the half of time slots.

The difference between the maximum and minimum number of idle time periods can be at most h.

$$dif_j \leftrightarrow min_j \, XOR \, max_j \qquad \forall j \in 1..\lfloor (nTimeSlots - 1)/2 \rfloor$$
$$atMost(h, \{dif_j \mid j \in 1..\lfloor (nTimeSlots - 1)/2 \rfloor\})$$

Implied Constraints: From the constraints defined for the problem, the following two sets of implied constraints on *usedSlot* variables are identified:

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.

$$exactly(|meetings(p)|, \{usedSlot_{p,j} \mid j \in \mathcal{T}\}) \;\; \forall p \in \mathcal{P} \tag{1}$$

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

$$atMost(2 \times nTables, \{usedSlot_{p,j} \mid p \in \mathcal{P}\}) \;\; \forall j \in \mathcal{T} \tag{2}$$

These constraints can be encoded in several ways into SAT, being *cardinality networks* [1] the most promising approach on this problem, according to the experiments in [5]. Moreover, as shown by the experiments, using those implied constraints is in general beneficial. However, it is difficult to extract conclusions from those experiments, due to the limited number and variety of B2B instances.

3 A Random B2B Instances Generator

In this section we present a model of generation of random B2B problems. To model these problems, we consider a number of participants $P = |\mathcal{P}|$ and a number of meetings $M = |\mathcal{M}|$. The key question is how these M meetings are distributed among these P participants. In this distribution, we need to impose a restriction to ensure the feasibility of the instance: the number of meetings requested by a participant cannot be greater than the number of time slots $T = |\mathcal{T}|$. Also notice that this distribution is independent on the number of locations $L = |\mathcal{L}|$. For simplicity, we do not consider either forbidden or morning/afternoon time slots.

The set of available real-world B2B instances [5][3] contains 20 instances: 5 of them come from real data, and the remaining 15 are crafted modifications from the real ones. As we cannot draw general conclusions about probability distributions from this reduced set of real instances, we present a simple model to generate random B2B instances. This model, called *regular model*,[4] is based on a probability U that a participant requests a meeting with another.

Definition 2 (Regular Model)**.** *Let P be a positive number of participants, and U a real value in $[0, 1]$. The probability that a participant p_i requests a meeting with another participant p_j is exactly U, where $1 \leq i, j \leq P$ and $i \neq j$.*

Theorem 1. *In the* regular model *for random B2B problems, the number of meetings requested per participant follows a binomial distribution $B(n, p)$ with parameters $n = P - 1$ and $p = U$.*

Proof. Each participant can request meetings with the remaining $P - 1$ participants. This means that for each of them, there is a sequence of $P - 1$ *yes/no* independent experiments, each of which yields success with probability U. This is exactly the definition of a binomial distribution $B(n, p)$ with parameters $n = P - 1$ and $p = U$. □

[3] Instances available at http://imae.udg.edu/recerca/lap/simply/
[4] In case of acceptance, this generator will be publicly available.

Corollary 1. *In the* regular model*, the expected number of meetings requested by each participant is* $(P-1)U$*.*

Proof. In a binomial distribution $B(n,p)$, the mean is np. □

Note that our model allows us to generate instances in which the number of meetings requested per participant is close to the maximum. The regular model is not parametric on the number of time slots T. Thus, when $(P-1)U \gg T$, there exists (with high probability) a number of participants requesting more meetings than available time slots, and hence it is not possible to schedule all of them. To avoid trivially unfeasible instances, we perform a preprocessing step limiting the maximum number of requests of a participant to T.[5] This way, the resulting problem contains many participants requesting T meetings, i.e., many observations close to the maximum. In our experience, we have found this kind of participant is very frequent.

In what follows, we use this model to generate random families of B2B instances, and we study the performance of the solving process on the use of the implied constraints. This study aims to check whether it is beneficial to use them or not, and in which situations is more useful to use one or another. In particular, we focus our study on two features of the problem: the *density* and the *shape*.

Definition 3 (Density of a B2B instance)**.** *Given a positive number of meetings* M*, time slots* T *and locations* L*, the density* d *of a B2B instance is the relation between the number of meetings* M *and the accommodation capacity* $T \cdot L$*.*

$$d = \frac{M}{T \cdot L} \tag{3}$$

Notice that if the density of a B2B instance is greater than one, the instance is necessarily unfeasible. The opposite is not true. Let us consider a B2B instance with 3 participants, 3 meetings between the 3 combinations of them, 2 time slots, and 2 locations. The density of this instance is $d = 3/4$. However, it is easy to see that this instance is unfeasible. In fact, increasing the number of locations, i.e., decreasing the density, does not modify its feasibility. Therefore, unfeasible instances with density smaller than one exist.

Definition 4 (Shape of a B2B instance)**.** *Given the accommodation capacity* $T \cdot L$*, the shape* s *of a B2B instance is defined as the relation between the number of time slots* T *and the number of locations* L*.*

$$s = \frac{T}{L} \tag{4}$$

Finally, we remark that this model does not represent all features of real-world instances. For instance, using this model we cannot generate a large number of participants requesting a number of meetings far from the mean. In particular, it may be possible the existence of *passive* participants in some B2B events. These participants are characterized by requesting no meetings (i.e, they attend the event because other participants request meetings with them). In this case, a polynomial decreasing distribution, as a power-law distribution, may model these instances more adequately. However, our model seems adequate to model B2B instances similar to the known real-world ones.

[5] A trivially unfeasible instance contains participants requesting more meetings than available time slots.

4 Experimental Evaluation

In this section, we evaluate the effect on the solver performance of the use of implied constraints. To do so, we first generate some families of random B2B instances, and we solve them with and without using implied constraints. The goal of these experiments is to identify those cases for which the use of some implied constraint is beneficial. Then, we validate these observations of random B2B instances in real-world problems. Finally, we conjecture the reasons of the success of using implied constraints based on some observations from the solver.

4.1 Experimental setup

In our experiments, we generate families of random B2B benchmarks, containing each of them 20 instances per configuration. We use 16 different configurations, resulting in a total of 320 different random instances for our experimentation. All experiments are run with a timeout of 2 hours (7200 seconds). As in [5], we use a value for homogeneity $h = 2$. We solve each instance without using implied constraints, and with using implied constraint 1 (Eq. 1), 2 (Eq. 2) and both. In the plots, these methods are named as *no-imp*, *imp1*, *imp2* and *imp12* respectively. We use Open-WBO [9] as MaxSAT solver. This solver has been ranked as one of the best non-portfolio solver in the industrial partial MaxSAT tracks of the MaxSAT Evaluations 2014 and 2015.[6] Random B2B instances were solved in a cluster of nodes with 32GB of RAM and 2 processors Intel(R) Xeon(R) @ 2.27 GHz, limiting all experiments to a single core and to a maximum of 4GB of RAM. Real-world B2B problems were solved in a cluster of nodes with 8GB of RAM and 1 processor Intel(R) Xeon(R) @ 3.1 GHz.

4.2 Random B2B Instances

In our experimentation, we use an estimation of the number of meetings M, since this number is unknown *a priori*. From Corollary 1, we know than the expected number of requests per participant is $(P-1)U$. Therefore, the expected total number of requests is $E[R] = P(P-1)U$. Notice that we distinguish between requests and meetings, since a meeting can be produced by a single request or by two (both participants request a meeting with each other). Therefore, the expected number of meetings is bounded by:

$$E[M] \leq E[R] = P(P-1)U$$

However, when the probability U is small, the number of requests is also small, and hence the number of meetings can be approximated as $E[M] \approx E[R]$.[7] Finally, we approximate M using its expected value, i.e., $M \approx E[M]$. In what follows, we use these approximations. In our experiments, we set the number of participants $P = 40$. We consider this number big enough to reproduce the hardness of some real-world problems. Notice that the number of participants in some real instances ranges between 42 and 47 (see the `tic` family in [5]). Also, we use a fixed probability $U = 0.1$. With the previous approximation, and using $P = 40$ and $U = 0.1$, we obtain $E[M] = 156$, which is a reasonable number of meeting w.r.t. real instances (whose number of meetings ranges between 125 and 184 for the previously mentioned family).

Based on the definitions of density and shape, we can easily create families of random B2B problems using the regular model and just modifying the input values of T and L. The natural

[6] http://www.maxsat.udl.cat/.

[7] The expected number of meetings is exactly $E[M] = \binom{P}{2}(2U(1-U)+UU) = \frac{P(P-1)(2U-U^2)}{2}$, which is the probability that one or both participants of a meeting request it, among the total pairs of participants. When U is small, we can approximate it as $E[M] \approx P(P-1)U$.

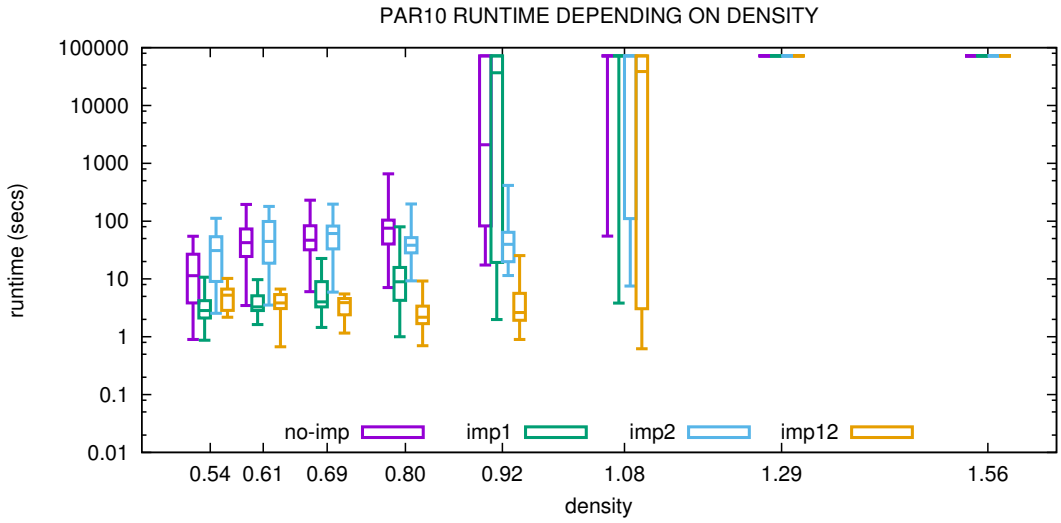

Figure 1: PAR10 (in seconds) of solving some random B2B families of instances, with and without using implied constraints, varying their density d. Instances are generated using the regular model with $P = 40$, $U = 0.1$, and a fixed shape $s = 1$ (i.e., $T = L$), hence $d = M/TL \approx P(P-1)Us/T^2$.

question now is to detect those cases in which the use of some implied constraint is more beneficial than any other encoding. To find those cases, we generate families of random B2B instances varying the density and the shape.

In our analysis, we represent the Penalized Average Runtime 10 (PAR10), which is the average of the runtime used to solve the set of instances, assigning to those instances with timeout (i.e., 7200 seconds) a runtime equal to 10 times the value of the timeout (i.e., 72000 seconds). For each family, we represent a box-and-whisker plot, which represents the maximum, minimum, median, and quartiles 1 and 3 of the runtimes of the family.

In Figure 1, we represent the runtime of solving some families of random B2B instances varying the density d, with a fixed shape $s = 1$ (i.e., $T = L$, and $P = 40$, $U = 0.1$). Recall that we approximate the density d as follows:

$$d = \frac{M}{T \cdot L} \approx \frac{P(P-1)U}{T \cdot T/s}$$

Notice that when P, U and s are fixed, the variations in the density are equivalent to variations in T (or L). First, we observe that denser the instance, harder to solve. This happens for all the 4 encodings. For instance, when $d = 1.29$ ($T = 11$) no instance is solved for none of the encodings, but they all solve all instances when $d = 0.54$ ($T = 17$) in approximately less than 100 seconds. Second, we observe that, in general, using no implied constraints (*no-imp*) is slower than using one implied constraint (either *imp1* or *imp2*), and using one is slower than using both (*imp12*). This was already suggested in [5]. Third, we observe that, interestingly, *imp2* shows a good performance w.r.t. *no-imp* when the density is high. For instance, when $d = 1.08$ ($T = 12$), *imp2* solves a total of 8 instances (40% of the family) while *no-imp* (and also *imp1*) only solves 2 of them (10%). On the contrary, when the density is small, it is *imp1* which shows a better performance w.r.t. *no-imp*. For instance, when $d = 0.54$ ($T = 17$), the

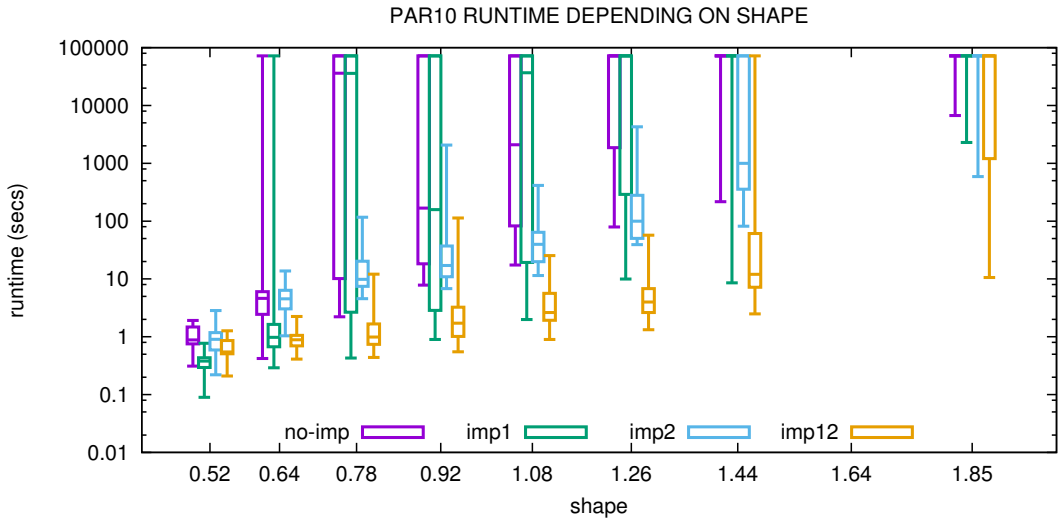

Figure 2: PAR10 (in seconds) of solving some random B2B families of instances, with and without using implied constraints, varying their shape s. Instances are generated using the regular model with $P = 40$, $U = 0.1$, and a fixed density $d \approx 1$ (i.e., $T \cdot L = E[R]$), hence $s = T/L \approx T^2/P(P-1)U$.

maximum runtime of *imp1* is 10.69 seconds, while *no-imp* spends a maximum of 54.75 seconds (the maximum of *imp2* is 111.85 seconds). We summarize these results in Observation 1.

Observation 1. *Using the implied constraint 1 (*imp1*) seems to be more interesting when the density is small. On the contrary, the implied constraint 2 (*imp2*) seems to be more relevant when the density is high. Overall, using both implied constraints (*imp12*) is always beneficial.*

In Figure 2, we represent the runtime of solving families of random B2B instances varying the shape s. Again, we set $P = 40$ and $U = 0.1$. We want to fix a density d. Originally, we used $E[R]$ to calculate T and L as follows: $T = L = \left\lceil \sqrt{E[R]} \right\rceil$. Therefore, for this P and U, T and L originally had a value of 13. In order to *fix* a density, the product of T and L should also be *similar*[8] to that number (169). Notice that this way, some combinations of (T, L) does not exist (e.g., $T = 16$). Therefore, the density d is close and slightly smaller than 1, and approximately the same for all families. Hence, the variations in the shape are also equivalent to variations in T (when P, U and d are fixed). In this experiment we identify several phenomena. First, the higher the value of the shape s (i.e., more time slots w.r.t. locations), the harder to solve the instance. Notice that for a fixed density, when the shape is *small*, the number of time slots is also *small* and the number of locations is *high*. Second, again *no-imp* is worse than *imp1* and *imp2*, and these two are worse than *imp12*, as suggested in [5]. Third, the benefits of using one of the implied constraints depend on the shape. When the shape is small, *imp1* is faster than *imp2*. For instance, when $s = 0.52$ ($T = 9$), *imp1* spends a total of 7.95 seconds in solving all instances of the family, while *imp2* spends 20.23 seconds. On the contrary, when the shape is high, *imp2* is more beneficial. For instance, when $s = 1.44$ ($T = 15$) *imp1* only solves 2 instances (10% of the family) while *imp2* solves 14 (70%). Finally, *imp12* is the best encoding,

[8] Notice that the T and L are integers, so an equal size may not be possible.

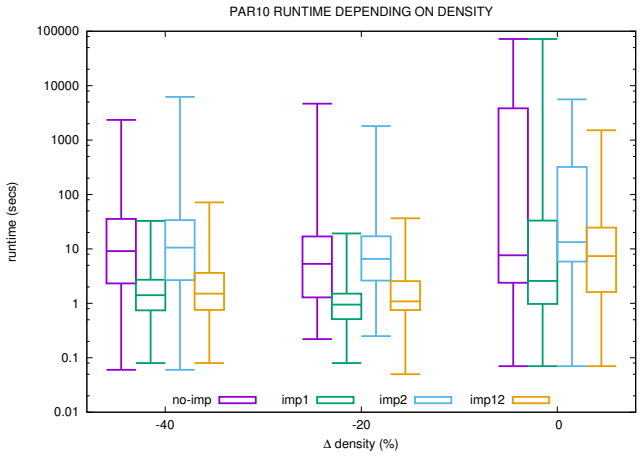

Figure 3: PAR10 (in seconds) of solving some real-world B2B instances, with and without using implied constraints, varying their density d, and with a fixed shape, i.e., $\Delta T = \Delta L$.

and harder the instance, better its performance w.r.t. the other 3 encodings. For instance, when $s = 1.85$ ($T = 17$), *no-imp*, *imp1* and *imp2* solve only 1 instance, while *imp12* solves 9. We summarize these results in Observation 2.

Observation 2. *Using the implied constraint 1 (*imp1*) seems to be more interesting when the shape is small. On the contrary, the implied constraint 2 (*imp2*) seems to be more relevant when the shape is high. Overall, using both implied constraints (*imp12*) is always beneficial.*

The intuition behind these observations may be connected to the relation between the implied constraints and the number of locations and time slots. In particular, the first implied constraint depends on the number of meetings of each participant, and this number is bounded by the number of time slots. The second implied constraint depends on the number of locations. Notice that since we are using cardinality networks to encode these constraints, the smaller the number of time slots is, the better for the first implied constraint encoding, and similarly with the number of locations and the second implied constraint.

4.3 Real-world B2B Instances

We want to check if the previous observations are also valid in real-world B2B instances. Notice that in the case of real-world instances, the combinations of T and L are reduced (in order to obtain feasible, non-trivial instances). Therefore, the number of perturbed problems (from the original ones) is smaller.

In Figure 3 we represent the runtime of solving some real-world B2B instances varying the density d, with a fixed shape s. To fix the shape of these problems, we use the original numbers of time slots and locations, and we increase both of them in the same proportion. Based on Observation 1, we predicted that *imp1* is more beneficial with small densities, *imp2* for a high density, and *imp12* shows a good performance in all cases. From the results, we observe that this observation is also valid in our set of real-world instances. For instance, when $\Delta d = -40\%$, the fastest encoding in solving all instances is *imp1*. Also, *imp2* solves all instances when $\Delta d = 0\%$ (in this case, there are several timeouts for *no-imp* and *imp1*). Finally, *imp12* is, in

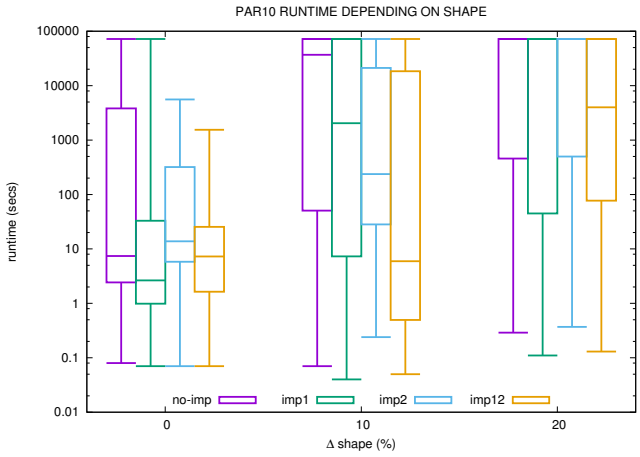

Figure 4: PAR10 (in seconds) of solving some real-world B2B instances, with and without using implied constraints, varying their shape s, and with a fixed density, i.e., fixed $T \cdot L$.

	$\Delta d = -40\%$			$\Delta d = -20\%$			$\Delta d = 0\%$		
	solved	PAR1	PAR10	*solved*	PAR1	PAR10	*solved*	PAR1	PAR10
no-imp	**20**	148.49	148.49	**20**	340.38	340.38	16	1835.82	14797.82
imp1	**20**	**3.54**	**3.54**	**20**	**3.48**	**3.48**	18	735.39	7216.33
imp2	**20**	470.21	470.21	**20**	190.59	190.59	**20**	746.34	746.34
imp12	**20**	7.13	7.13	**20**	6.12	6.12	**20**	**148.54**	**148.54**

Table 1: Statistics of solving some real-world B2B instances, varying their density d, for a fixed shape s (with $\Delta T = \Delta L$).

general, one of the best choices. This last claim is not clear from the plot. In Table 1, we report some statistics of this experiment. We use the Penalized Average Runtimes PAR1 and PAR10. Notice that the penalization in PAR1 is *small*, thus it is useful to compare the runtime of solving families where the majority of instances were solved. On the other hand, using PAR10 specially penalizes those timeouts, and thus it is useful when many instances were not solved. From these results, we observe that *imp12* is the fastest method in solving hard instances (see $\Delta d = 0\%$), but it is also a good choice when the instances are easy (see $\Delta d = -40\%$). In this last case, the differences between *imp1* (the fastest method) and *imp12* are very small. Therefore, this observation also seems to be valid in our benchmarks.

In Figure 4 we represent the runtime of solving some real-world B2B instances varying the shape s, with a fixed density d. To do so, we increase/decrease T and L in the same proportion.[9] The prediction from Observation 2 is that *imp1* seems to be more useful than *imp2* for small shapes, and vice-versa, while *imp12* is always beneficial. This claim is not totally observed from the plot, due to the number of timeouts. In Table 2, we report some statistical results. In this case, we consider more appropriate to use PAR1 in the cases of a reduced number of timeouts. We observe that PAR1 of *imp1* is smaller than PAR1 of *imp2* for small values of Δs (see $\Delta s = 0\%$). However, as ΔT increases, *imp2* is faster than *imp1*. Finally, we can observe

[9] For distinct values of T and L, increasing/decreasing them in the same proportion does not ensure that its product continues being the same. However, this does happen in our real-world instances.

	$\Delta s = 0\%$			$\Delta s = 10\%$			$\Delta s = 20\%$		
	solved	PAR1	PAR10	*solved*	PAR1	PAR10	*solved*	PAR1	PAR10
no-imp	16	1835.82	14797.82	10	3720.64	36125.48	5	5437.97	54045.19
imp1	18	735.39	7216.33	12	3281.89	29205.53	5	5397.89	54004.82
imp2	**20**	746.34	746.34	**15**	2289.69	18492.07	6	5142.26	50508.96
imp12	**20**	**148.54**	**148.54**	**15**	**1832.71**	**18035.06**	**10**	**4054.49**	**36458.95**

Table 2: Statistics of solving some real-world B2B instances, varying the shape s, for a fixed density d (with $T \cdot L$ fixed).

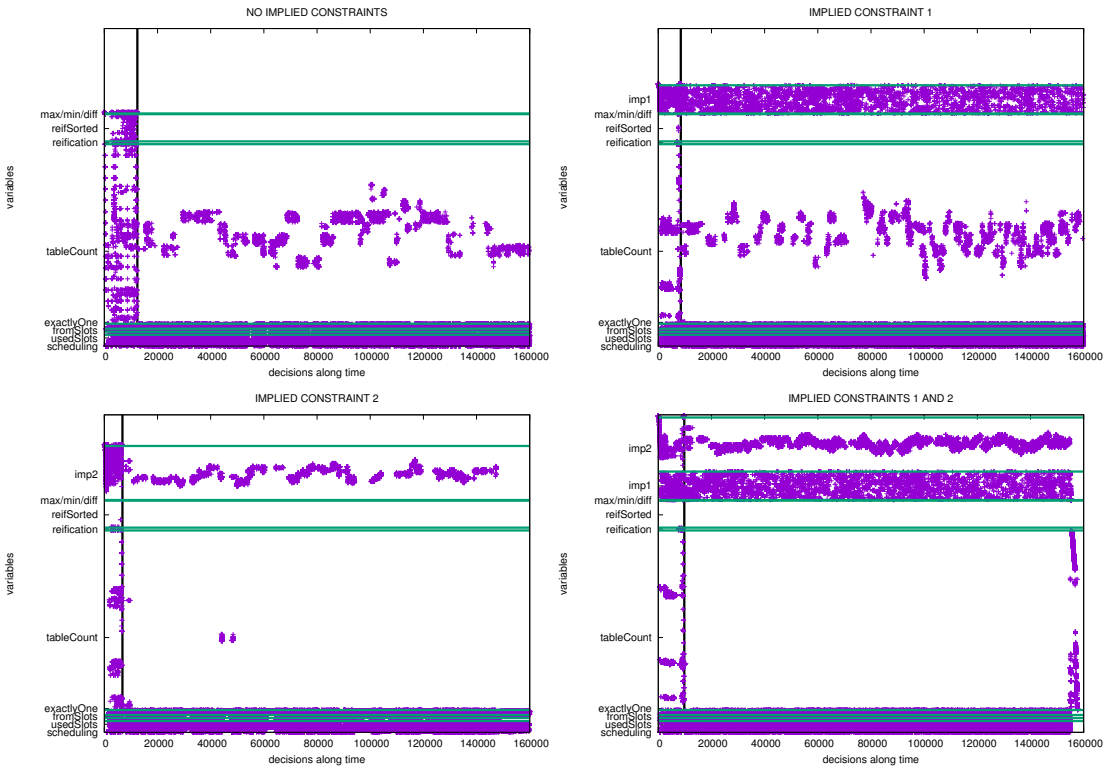

Figure 5: Branching variables decided by the solver along its execution, for a random B2B instance with low density, during their first 160000 decisions (solved by *imp12* in 157729 dec.).

than *imp12* dominates the other encodings. Therefore, this second observation also seems to be valid in our set of instances.

4.4 Performance of the MaxSAT solver

Finally, let us conjecture why the use of implied constraints is beneficial to improve the performance of the solver. In the following plots, all Boolean variables are grouped into the high-level variables and constraints they encode (see the horizontal lines). These high-level variables are: *scheduling*, *usedSlot* and *fromSlot* are directly the ones of the encoding, *exactlyOne* are the

auxiliary variables to encode that each meeting is scheduled in a time slot exactly once, *tableCount* are the auxiliary variables to encode that at most one meeting is scheduled in a time slot and location, *imp1* and *imp2* are the auxiliary variables to encode the implied constraints, and the rest are the auxiliary variables to deal with optimization and homogeneity. Vertical lines represent the calls to the SAT solver used by the MaxSAT solver algorithm[10].

In Figure 5, we represent the branching variables on which the solver decided along its execution, for the encodings *no-imp* (top left), *imp1* (top right), *imp2* (bottom left) and *imp12* (bottom right), for a random B2B instance generated with the regular model and low density ($P = 40$, $U = 0.1$, $T = 16$ and $L = 16$). For simplicity, we only represent the results of a single instance. However, we have found the same behavior in all instances we have analyzed. Therefore, we expect the conclusions drawn from this plot are *general*. According to Observation 1, instances with low density are solved faster by *imp1* (and *imp12*). In fact, the runtime and the number of decisions of each encodings are:

No implied constraints:	70.78 s	3444288 dec.
Implied constraint 1:	5.88 s	343487 dec.
Implied constraint 2:	84.16 s	3152049 dec.
Implied constraints 1 and 2:	3.07 s	157729 dec.

As this instance is solved by *imp12* in 157729 decisions, we only represent the first 160000 decisions for the 4 encodings. However, the behavior shown in the plots is the same during the whole execution.

We remark that Open-WBO uses a CDCL SAT solver internally (we use its default version, which uses Glucose 3.0). In CDCL solvers, after each conflict, the variables involved in it are increased their activity, and these activity counters are used by the branching heuristics to select the next decision. Therefore, these decisions give an intuition about where the search was performed.

We observe that in three encodings, variables *scheduling*, *usedSlot*, *fromSlot* and *exactlyOne* are very active during the whole execution. This is expected since they represent the most important variables of the problem. When we use one implied constraint only, this implied constraint is very active. Also, when we use both implied constraints, they reinforce their activity mutually, and hence, the performance of the solver is improved. This phenomenon is clear in the plot, but we can analyze it in details during the whole execution. For instance, in the encoding *imp1*, 3.90% of the decisions correspond to this implied constraint. When using the encoding *imp12*, the percentage of decisions in the variables of the implied constraint 1 is 6.32% of the total. Therefore, the use of *imp2* reinforces the activity of *imp1*, and hence, the instance is solved faster.

In Figure 6, we represent the results of the same experiment using an instance with high density ($P = 40$, $U = 0.1$, $T = 12$ and $L = 12$). Again, we only represent results for a single instance but conclusions are general for the family. According to Observation 1, instances with high density are solved faster by *imp2* (and *imp12*). This is also the case of this instance, whose runtime and number of decisions are:

No implied constraints:	74.47 s	1568888 dec.
Implied constraint 1:	36.97 s	631983 dec.
Implied constraint 2:	4.44 s	237389 dec.
Implied constraints 1 and 2:	0.84 s	42550 dec.

In this case, the encoding *imp2* took 12.44% of its decisions on the Boolean variables of this implied constraints. When we use both implied constraints *imp12*, the decisions on the variables of the second implied constraint represent the 17.86% of the total number of decisions.

[10] The MaxSAT solver at hand has solved all instances using the MSU3 algorithm, an UNSAT-based MaxSAT algorithm.

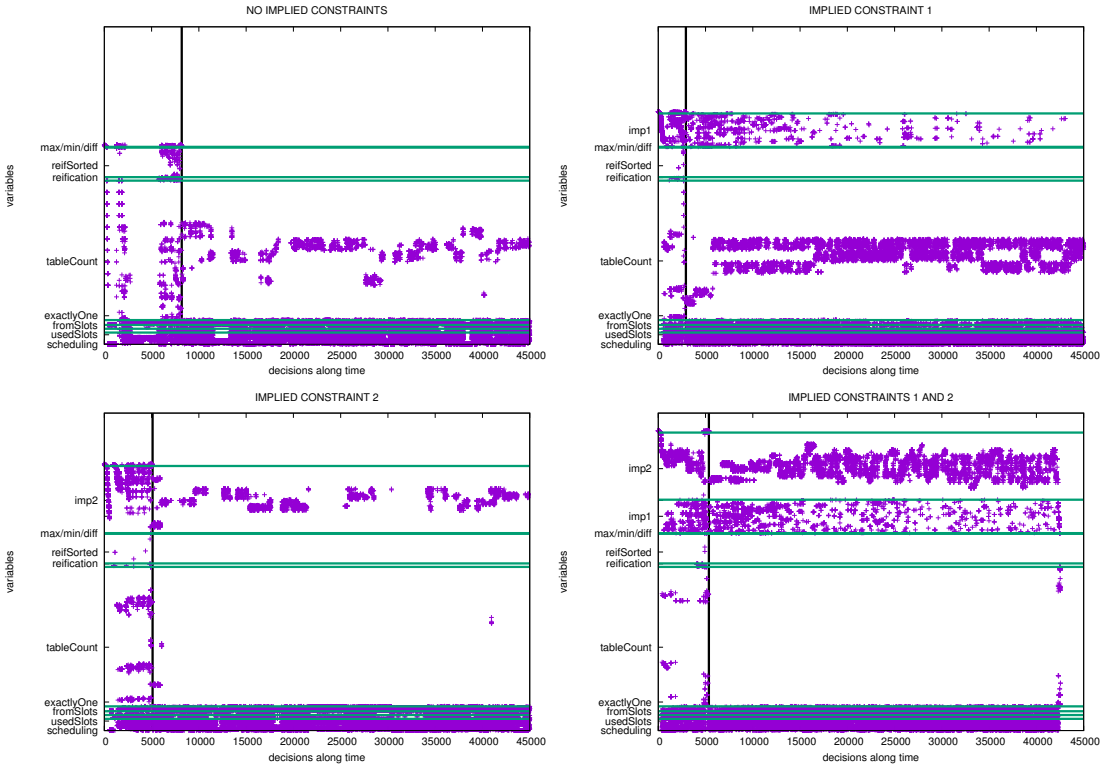

Figure 6: Branching variables decided by the solver along its execution, for a random B2B instance with high density, during their first 45000 decisions (solved by *imp12* in 42550 dec.).

We also check these effects on some real instances (see [5] for more details). The encoding *imp1* is more efficient for the instance `forum-14`, taking 2.54% of its decision on this constraint. When using both implied constraints, this increases up to 3.55% of the decisions. Similarly, the encoding *imp2* is more efficient solving the instance `tic-14crafd`, with a 4.93% of decision on this constraint. When using both implied constraints, these decisions are the 5.18%. This suggests that the previous hypothesis is also valid in real-world B2B problems.

Finally, we want to analyze the effect of implied constraints between two calls to the SAT solver (vertical lines in the previous plots). Notice that all previous random instances have an optimum equal to zero. This means that all meetings can be scheduled without idle periods. On the other hand, many real-world B2B instances do have these idle periods, i.e., their optimums are greater than zero, and hence, the SAT solver is called multiple times. Recall that the algorithm MSU3, used by the MaxSAT solver Open-WBO, first solves the formula only containing hard clauses. Then, it iteratively increases the possible number of unsatisfied soft clauses (i.e., a lower-bound of the optimum), till finding the solution of the problem. Therefore, solving any B2B instance requires at least two calls to the SAT solver.

Interestingly, we observed that the use of implied constraints allows to the SAT solver to find lower-bounds to the optimum with unit propagation. In particular, all calls to the SAT solver except the first and the last ones were solved without performing any decision. We conjecture that the existence of idle periods may be produced in some cases by the existence of forbidden

time slots (by a certain participant), and the use of implied constraints may detect them by unit propagation. On the contrary, this does not occur when no implied constraints are used. This suggests that redundant clauses (e.g., implied constraints) may increase the unit propagation rates, and hence, improve the performance of the solver.

5 Conclusions and Future Work

In this paper, we provide an experimental study of the effectiveness of using implied constraints in B2B scheduling problems using MaxSAT-based encodings.

Due to the limited number of real-world instances, we propose a random B2B instances generator. This model is based on a probability that a participant requests a meeting with another. Using our generator, we generate families of random B2B instances, and we study the strengths and weakness of using implied constraints based on the characteristics of the instance. We focus our analysis on the density (i.e., the ratio between the number of meetings and the accommodation capacity) and the shape (i.e., the configuration of the accommodation capacity).

We observe that there exists a duality in the benefits of the use of the two implied constraints studied in this work. For small densities or certain shapes, it is more useful to use one of these implied constraints. On the contrary, for high densities or opposites shapes, the other implied constraint is more beneficial. Overall, the use of both implied constraints results into a very good performance in all cases. Finally, we conjecture why this is the case by some observation of the solver. We illustrate how the solver focuses its search when both implied are used.

As future work, we propose to analyze heuristics that, for instance, prioritizes its decisions on the most relevant variables we have analyzed (i.e., *imp1* or *imp2*), depending on the characteristics of the problem, as the density or the shape.

References

[1] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In *19th International Conference on Principles and Practice of Constraint Programming , CP 2013*, volume 8124 of *LNCS*, pages 80–96. Springer, 2013.

[2] Teresa Alsinet, Ramón Béjar, Alba Cabiscol, Cèsar Fernández, and Felip Manyà. Minimal and redundant SAT encodings for the all-interval-series problem. In *5th Catalonian Conference on Topics in Artificial Intelligence CCIA 2002*, pages 139–144, 2002.

[3] Carlos Ansótegui, Alvaro del Val, Iván Dotú, Cèsar Fernández, and Felip Manyà. Modeling choices in quasigroup completion: SAT vs. CSP. In *19th National Conference on Artificial Intelligence, AAAI 2004*, pages 137–142, 2004.

[4] Miquel Bofill, Joan Espasa, Marc Garcia, Miquel Palahí, Josep Suy, and Mateu Villaret. Scheduling B2B Meetings. In *20th International Conference on Principles and Practice of Constraint Programming, CP 2014*, volume 8656 of *LNCS*, pages 781–796. Springer, 2014.

[5] Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. MaxSAT-Based Scheduling of B2B Meetings. In *12th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR 2015*, volume 9075 of *LNCS*, pages 65–73. Springer, 2015.

[6] Martin Gebser, Thomas Glase, Orkunt Sabuncu, and Torsten Schaub. Matchmaking with answer set programming. In *12th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, pages 342–347, 2013.

[7] Henry A. Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla P. Gomes, Bart Selman, and Mark E. Stickel. Balance and filtering in structured satisfiable problems. In *17th International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 351–358, 2001.

[8] Chu Min Li and Felip Manyà. *Handbook of Satisfiability*, chapter MaxSAT, Hard and Soft Constraints, pages 613–631. IOS Press, 2009.

[9] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver,. In *17th International Conference on Theory and Applications of Satisfiability Testing, SAT 2014*, pages 438–445, 2014.

[10] Gilles Pesant, Gregory Rix, and Louis-Martin Rousseau. A Comparative Study of MIP and CP Formulations for the B2B Scheduling Optimization Problem. In *12th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR 2015*, volume 9075 of *LNCS*, pages 306–321. Springer, 2015.

[11] Bart Selman, Henry A. Kautz, and David A. McAllester. Ten challenges in propositional reasoning and search. In *15th International Joint Conference on Artificial Intelligence, IJCAI 97*, pages 50–54. Morgan Kaufmann, 1997.