Analysis of a MaxSAT approach of the B2B Scheduling Optimization Problem.

Marc Garcia
Supervised by: Dr. Mateu Villaret

Universitat de Girona, Spain
nikano1991@gmail.com

Contents

1 Introduction

Business-to-business (B2B) are events where brief meetings between participants with similar interests are celebrated. These participants can be companies, investors, research groups, etc. That kind of events come up in several fields like sports, social life, research, etc.

A close example can be found in the *"Parc científic i tecnològic de la UdG"*[1] where every year two of these events were hold. They were the *"Forum del Parc científic i tecnològic de la Universitat de Girona"*[2] and the *"Jornades R+D+I en Tic i Salut"*[3].

In this project we deal with the problem of scheduling the meetings of a B2B event. This problem, known as the B2B Scheduling Problem, and its optimization variation, the B2B Scheduling Optimization Problem was firstly presented in [10] and in [6].

To obtain a schedule of the meetings there are two stages. The first one consists on getting and filtering all the requested meetings, for this job, it is necessary a human expert who can choose appropriately witch meeting can (or must) be discarded, this expert is the human matchmaker. This stage will not be considered in this project as widely as it is done in [15], and mainly we will assume that this is already done. Once this first pruning of meetings is done, what we have will be the final set of meetings that will appear on the final schedule.

The second stage is when the schedule is actually done. This schedule must satisfy some constraints, e.g., avoid meetings collisions, avoid unnecessary idle times between meetings for each participant, minimize the number of meeting location changes, etc.

Previously to the work done in this master thesis, B2B have been considered in several works: CP,TFG, AsP. Here we improve the MaxSAT encoding of [15] using cardinality constraints [8] and proposing two implied constraints that dramatically improve solving time for several instances and propose some symmetry breaking with almost no effect.

In a second part of the thesis we try to identify the reason of the good performance obtained thanks to those implied constraints. Namely we study the benefits of using implied constraints with respect the density and the

[1]`http://www.parcudg.com`
[2]`http://www.forumparcudg.com`
[3]`http://www.jornadaticsalut.com`

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. To do that we also provide an instance generator. This second part has resulted in the publication [7]. To be complete w.r.t. related work, in [16] there are some alternative CP and MILP based approaches to B2B solving. Our MaxSAT proposal clearly dominates the solving time in all instances, hence it is state-of-the-art.

For the events celebrated on the *"Parc científic i tecnològic de la UdG"*, the schedule of the meetings had been done manually for a human expert. The problem was that in these events there are about three hundred requested meetings if not more. These big numbers made the task of schedule them very tough to be done manually.

A automatized process have been used since it was presented in [6]. Certainly the use of this method have improved greatly the handmade solutions in terms of time to obtain the solution, and quality of the solution.

1.1 Goals

This projects can be divided into three main goals.

1. Improve the MaxSAT implementation of the problem that was presented in [15]. To accomplish this we set the following objectives:
    - Identify the flaws in the current version of the MaxSAT implementation.
    - Encode the cardinal constraints of the problem using the state-of-the-art techniques presented in [1] [4].
    - Identify possible implied constraints that may improve solving time.
    - Compare the different encodings with and without implied constraints with each other and with the previous MaxSAT model for the twenty instances of the problem.
2. Create an instance generator. The objectives are the following:
    - Use the four reals instances of the B2B scheduling problem to find some patterns that can be use to model an instance generator.

   - Model the B2B instance generator.
   - Make the B2B instance generator parameterizable in several ways in order to reproduce certain characteristics of B2B real-world problems.

   The development of random problems generators sharing the majority of real-world instances features was already stated on [17] as one of the most important challenges of the next years.

   Notice that this it is not a goal by itself, but a mean to be able to carry on with the third goal.

   This generator will allow us to augment the set of problems since it is expected that the random generated instances will share the main characteristics of the known problems. Having this generator will also allow us to create instances of any desired size.

3. Study the use of impled constraints. The objectives of this part are the following:

   - Identify different implied constraints for the proposed MaxSAT model.
   - Define some families of instances to be used to study the behaviour of implied constraints.
   - Compare the use of implied constraints on the different families defined. This instances will be obtained by means of the instance generator.
   - Check if the behaviour seen when using implied constrains on the generated instances also happens with the real instances of the problem.
   - Illustrate how the use of implied constraints affects the solver decisions, and explain its possible relation with the solver good performance when using them.

1.2 Research group LAP

I would like to thanks the *Logic and Programming research group (LAP)*[4] which is a research group of the University of Girona (UdG). They have

[4] `http://ima.udg.es/Recerca/GrupESLIP.html`

been helping and supporting me during the elaboration of this project.

The research group is mainly focused in propositional satisfiability (SAT), SAT modulo theories (SMT) and their applications to solve combinatorial problems in areas such as planning and scheduling: timetabling, task sequencing in industrial processes, etc.

Before starting this project, I did some work experience in the research group supervised by Mateu Villaret[5]. It was also supervised by him and with the support of the whole group that I did TFG (*"Treball Fi de Grau"*) [15]. During this time working with them, we have presented three papers related with the B2B scheduling problem [6] [8] [7] to different well considered congresses.

1.3 Sections of the project

This is a research project and it means that the structure of the sections of the project may differ from other projects. I start the Section 1 , *Introduction*, with a little explanation of the problem at hand, and stating the goals of this project. After that it follow with Section 2, *Viability*, where I explain the things that have made this project possible. Then, in Section 3, *Methodology*, I explain the extreme programming methodology which is the methodology used. The next Section is 4, *Planning*, where I show how the project was planned, and how it changed over time because of new requirements, dead roads, etc.

It is followed by the Section 5, *Prior knowledge*, where some important concepts to be able to fully understand the work done here are given (e.g., NP Problem, Satisfiability problem, Implied constraints). Next there is the Section 6, *Requirements and decisions*, where I comment the software and hardware requirements of the project.

In Section 7, *Definition of the problem*, a fully detailed explanation of what B2B events are and the B2B problems is given. In Section 8, *Implementation*, I present the improved MaxSAT encoding and the random B2B instance generator. After that, in Section 9, *Results*, I show and analyze the results obtained from the comparison of the MaxSAT models and from the study of the implied constraints.

Finally in Sections 10, *Conclusions*, and 11, *Further work*, I sum up the work that has been done in this project and I open some roads to continue

[5]`http://ima.udg.es/~villaret/`

studying the B2B scheduling problem.

2 Viability

This project has been possible to do thanks to the research group LAP, the *"Parc cientific i tecnològic de la UdG"*, and the previous work done by myself in Scheduling B2B meetings [6] [15].

The LAP group is a research group of the University of Girona which has experience with scheduling and timetable problems, I was able to pick up the knowledge that was needed to develop this project.

In addition, the group has a cluster formed by thirteen machines which they put at my disposal. It was really useful to have these machines since because of the nature of this project I needed to run lots of experiments, and this way I was able to do this more quickly and with machines with the same features, which made it more fair when it came to benchmarking.

There also is the *"Parc científic i tecnològic de la UdG"* which were the ones that brought the problem to the research group LAP asking for an automatized method for getting the meetings timetable. It was because, in a few months time, they would hold the event *"Forum del parc científic i tecnològic de la Universitat de Girona"* in which it would become very useful to have that kind of automatized process.

The *"Parc científic i tecnològic"* provide us with a lot of help during the development process. They also put us in touch with Mireia Centelles, which was the person in charge of the project, and she also did the job of the *human expert.*. Mireia helped us in a different ways, e.g., giving us the criteria to make the filtering of the requested meetings, verifying that the solution given by the software was correct, etc.

They also gave us the requested meetings of the previous editions of the event, so we could use them to test our software and compare it with the handmade solutions.

Finally I would like to put emphasis in the work related with the B2B scheduling problem that was already done when I started with this project. The work that I am talking about is the one called *Scheduling B2B meetings* [15] done by myself and presented in 2014. From that project also came out the paper *Scheduling B2B meetings* [6] presented on the *The 20th International Conference on Principles and Practice of Constraint Programming* also in 2014. It was there and in [10] where the B2B scheduling problem was formally defined in the literature for the first time.

The work done in 2014 can be seen as a first approach to the B2B problems, it is focused with the whole process, since the moment that the participants make their requests until the very final schedule. In there, several different models for the the B2B Scheduling Problem (B2BSP), its optimization variation, the B2B Scheduling Optimization Problem (B2BSOP) and the B2B Location Optimization Problem (B2BLOP) are presented and compared between them. From that work it came clear that a very interesting way to approach the B2BSOP (which is the most interesting from of the three) was doing a MaxSAT encoding.

The project that I am presenting here is the one that comes after having get our hands dirty with the B2BSOP, having understood its behaviour, and being able to provide a much more competitive MaxSAT model that solves the problem. Here we also do a deep study of the use and the impact of implied constraints.

3 Methodology

In order to develop this project, since it is a research project, and not a software project, we decided that the methodology that best fits was the *Extreme programming*[6]. Notice that the methodologies are major focalized for software development as it, and they are hard to adapt when it comes to research.

The choice of useing the *Extreme programming* methodology (or a variation of it) was because of the fact that when doing research it is highly possible to find oneself into a dead end. When this happens it is time to go back a few steps, take into consideration all the problem as a whole and try another way. This process can be seen as a change on the requirements of the software, and the *Extreme programming* methodology is a very good one when it is predictable to have lots of changes.

Also, when doing research, it is a good approach to start getting results as quick as possible and then iterate over the implementation in order to improve the results. It fits perfectly with the *Extreme programming* philosophy which says that the most important thing is the code.

3.1 Extreme programming

Extreme programming is a software development methodology of the type of *agile software development*. Because of this, it is based in short development cycles which are intended to improve productivity and to offer the possibility of adding new requirements.

Extreme programming is based in four principles, these are the following:

- Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentations. Extreme programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team.

- Extreme programming encourages starting with the simplest solution. Extra functionality can be added later. The difference between this approach and more conventional system development methods is the

[6]`https://en.wikipedia.org/wiki/Extreme_programming`

focus on designing and coding for the needs of today instead of those of tomorrow. This is something known as the *"You aren't gonna need it" (YAGNI).*

- Feedback comes from different sources:
  - Feedback from the system: by writing unit tests. It provides to the programmers with direct feedback from the sate of the system after implementing changes.
  - Feedback from the customers: functionality test are shown to the client. It give to the client information about how is the software implementation going and helps them to decide if it is necessary to add new requirements.

  Usually in the research wold there is not the figure of the client, and it may be seen as a very self demanding developer.

- Extreme programming argue that only truly important product of the system development process is code. Codding can also be used to figure out the most suitable solution.

4 Planning

In this section we present all the process that we planned to do in this project, as well as the estimated time for each task, the dead ends that we found while doing the project and also the things that went well.

As it was said before, this project has three goals.

The first one is to improve the existent MaxSAT model of the B2B Scheduling Optimization Problem. Here we want to compare the new model with the old one and see how much we were able to improve.

The second goal is to create an instance generator. Here what we want is to be able to have a huge set of instances for the B2B scheduling problem so the next goal can be done more accurately. Notice that this goal is only a mean to be able to accomplish the next goal.

The third goal is to make and exhaustive study of the use of implied constraints. Here we want to see if the implied constraints work well for real instances, and if so, why they do and how they effect the solver decisions.

In order to accomplish the three goals, this project was planned as follows:

1. First of all, we book some time to do a deeper study of the B2B Scheduling Optimization Problem and also to see the state-of-the-art SAT encodings. The goal of this study was to find what and how the MaxSAT model and the encodings presented in [15] could be improved.

   For this we estimated two weeks of investigation.

2. After having done some research, we found three major points where the MaxSAT model could be improved.

   - Change the encoding of the cardinal constraints. Currently we were using a very simple implementation of the cardinal constraints, and since there are lots of these constraints in our problem we planned to change the naive encodings that we where using on [15] to a state-of-the-art encodings [1] [4].
   - Break symmetries. Symmetries are parts of the search tree that do not need to be explored since they can be deduced from some others branches of the tree with a linear operation. Break symmetries means to stop the search of these useless branches. It usually yields some good improvements so we planned to search and add some constraints to the model to break symmetries.

- Use implied constraints. Implied constraints are constraints that do not modify the set of solutions nor the search tree, but to use them usually means reduce the solving time. We also planned to search some implied constraints and implement them.

The process of decide what to do next took about a week.

3. We started with the new implementation of the cardinal constraints.

   This implementation was estimated in three weeks

4. Once the new implementation was finished we use the twenty instances that we had to run our experiments and compare the results with the old implementation. Here we saw that using cardinal networks for the cardinal constrains improved our solving time a lot.

   This comparison was estimated with half a week.

5. The next step was the find and implement symmetry breaking. This part proved to be more difficult of what it was expected. At the end the results that we obtained where quite a deception, since they did not improve the solving time, and in some cases they even made it worse.

   It was estimated in two weeks but it took almost a month.

6. We considered the symmetry breaking a dead end.

7. We started with the search for implied constraints. We were able to find two implied constraints which we implemented.

   This was estimated in two weeks.

8. Once we had the new implementation with the implied constraints we use again the twenty instances that we had to run our experiments. The results that we obtained using implied constraints were really interesting.

9. At this point, and since we had obtained some competitive results, we write a paper to be send to the twelfth international conference on Integration of *Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming* [7] [8].

   It took two weeks.

[7] `http://cpaior2015.uconn.edu/`

10. Considering the interesting results obtained with the implied constrains we decided to investigate this behaviour. It became obvious though that if we wanted to do that we needed a bigger set of instances.

11. We decided to create our own instance generator for the B2B scheduling problem, since it was (almost) impossible to obtain more real instances for it.

12. For creating the instance generator it was necessary to study the patters of the four industrial instances that we had and implement it.

    This process was estimated with a week.

13. Sadly it was not possible to extract useful information using only four instances, and the instance generator that we created was not useful. At the end we decided to create a random instance generator.

    This process took another week.

14. Once we had the set of *random* instances we could make a very exhaustive analysis of the behaviour of the solver when using implied constraints. This was estimated with two weeks.

15. With that analysis and with the instance generator we wrote another paper, this one to the annual *IEEE International Conference on Tools with Artificial Intelligence* [8].

16. This last paper was not accepted in the conference. However we took the advices of the reviewers and we also investigate a bit more the behaviour of the implied constraints especially the variable decisions which bring some interesting information.

17. With these new results we wrote a third paper, this time to *Pragmatics of Satisfiability* [9] [7] which is a workshop held within the international conference of satisfiability. There the paper was finally accepted.

The figure 1 shows an schematic representation of the planning of the project.

[8]`https://sites.google.com/site/ictai2015italy/`
[9]`http://www.pragmaticsofsat.org/2016/`

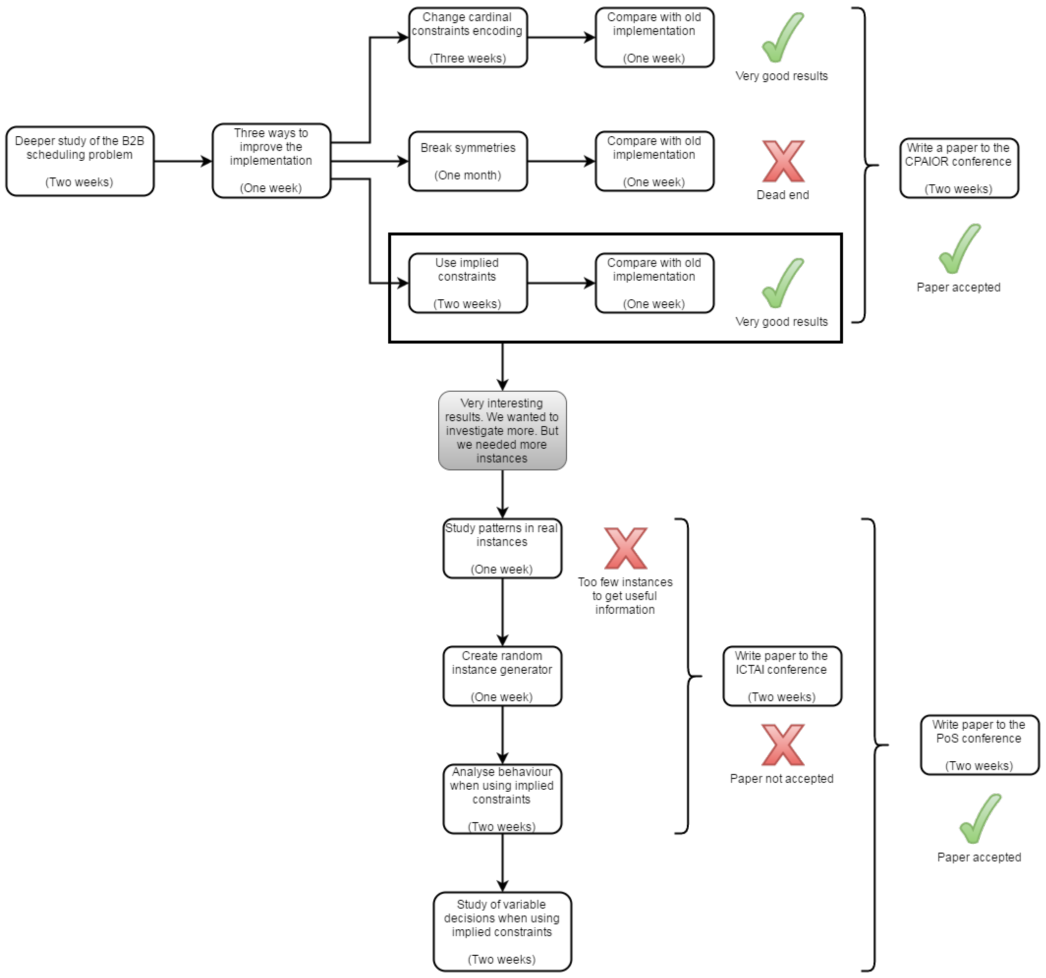
Deeper study of the B2B scheduling problem (Two weeks)
Three ways to improve the implementation (One week)
Change cardinal constraints encoding (Three weeks)
Compare with old implementation (One week)
Very good results
Break symmetries (One month)
Compare with old implementation (One week)
Dead end
Use implied constraints (Two weeks)
Compare with old implementation (One week)
Very good results
Write a paper to the CPAIOR conference (Two weeks)
Paper accepted
Very interesting results. We wanted to investigate more. But we needed more instances
Study patterns in real instances (One week)
Too few instances to get useful information
Create random instance generator (One week)
Analyse behaviour when using implied constraints (Two weeks)
Study of variable decisions when using implied constraints (Two weeks)
Write paper to the ICTAI conference (Two weeks)
Paper not accepted
Write paper to the PoS conference (Two weeks)
Paper accepted

Figure 1: Graphic representation of the planning of the project

5 Prior knowledge

Before starting to describe and analyse the implementation and the results, it is necessary to provide some insights about the technologies, techniques and other important concepts that will help to understand better the next sections.

5.1 NP problems

In computational complexity theory, on one hand we have the class of problems "P", *Polynomial time*, means that the problem can be solved on a deterministic sequential turing machine in an amount of time that is polynomial in the size of the input.

On the other hand-side there is the class of problems "NP", *Nondeterministic polynomial time* where you consider a non-deterministic turing machine instead.

Although any given solution for an NP problem can be verified in polynomial time, there is no known (nowadays) efficient way to find a solution for the problem. It means that the time required to solve a problem using any current known algorithm increases exponentially as the size of the problem grows.

Inside of the NP problems group can be found the so-called subgroup NP-complete problems. These are NP problems that have the characteristic that every NP problem can be reduced to them in polynomial time.

This leads us to one of the most important open questions in complexity theory, and this is the P = NP problem. This problem asks whether some polynomial time algorithms exists for NP-complete problems.

5.2 Constraint Satisfaction Problem (CSP)

Constraint satisfaction problems (CSP) are problems where the aim is to search for a state in witch a number of constraints are satisfied at the same time.

The constraints used in constraint programming are of different kinds: logical constraints (e.g. "A or B is true"), arithmetic constraints (e.g. "$x \leq 2$"), and many others.

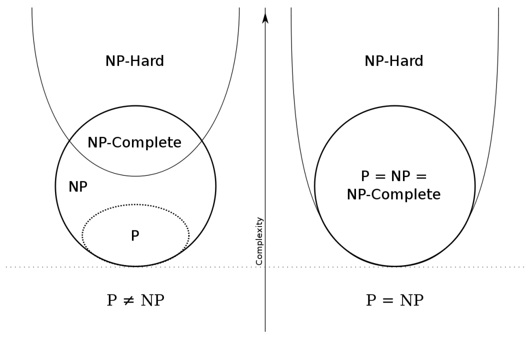

Figure 2: Euler diagram for P, NP, NP-complete and NP-hard set of problems

A variation of CSP is the Max-CSP where a number of constraints are allowed to be violated. In this case the quality of the solution is measured by the number of satisfied constraints. Another variation is the weighted CSP, which is a Max-CSP where some constraints have a weight w, and it means that if we violate some of these constraints, the solution will have a cost equal to the sum of all w from the violated constraints. Notice that it is possible to find a model problem that has, at the same time, the so-called hard and soft constraints. Hard constraints are the ones that must be satisfied always, while the soft constraints are the ones that have a w associated.

Boolean satisfiability problems (SAT), satisfiability modulo theories (SMT) and answer set programming (ASP) are different forms of the constraint satisfaction problem.

5.2.1 Satisfiability problem (SAT)

A boolean expression, e.g., $(A \vee B) \wedge (\neg C \wedge D)$, is a logic formula constructed from variables, operators AND, OR, NOT and parenthesis. A formula is satisfiable if it can be evaluated to $true$ by al least one appropriate assignation of logical values to the variables.

Thus, the boolean satisfiability problem (SAT) consists on determining if there is a model (an assignation of variables that make the formula evaluate as $true$) for a given boolean formula in conjunctive normal form.

If there is at least one model then the boolean formula is said to be satisfiable, otherwise it is unsatisfiable.

SAT was the first NP-complete problem known. As mention before, it means that any other NP problem can be reduce to the SAT problem, and most important, this reduction can be done in polynomial time. It is because of that that the SAT solvers (algorithms that try to solve SAT instances in a efficient way) have become more and more popular.

Thus, to solve a problem using SAT means to reduce that problem to SAT, and then send it to a SAT solver which will try to find a solution for that SAT instance. It is very important to keep in mind though that the SAT problem is an NP-complete problem, and despite of SAT solvers have improved a lot during the last years, there still is not any SAT solver that can solve all SAT instances in a polynomial time.

5.2.2 MaxSAT and weighted MaxSAT

MaxSAT and Weighted MaxSAT (WMaxSat) are the optimization variations of the SAT problem. Here the goal is to determine the maximum number of clauses of a SAT problem that can be made *true* at the same time by an assignment of boolean values to the variables of the formula. Let us take as an example the following boolean formula in conjunctive normal form:

$$(x_0 \vee x_1) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$$

The previous formula is known in the literature as a *clique*. A *clique* is a boolean formula where we have all the possible clauses that involve two variables, and it is trivially unsatisfiable. However, for the example at hand, it is possible to assign boolean values for the variables x_0 and x_1 in such a way that will make three out of the four clauses true. If this were a MaxSAT formula where each violation of a clause is penalized by 1 the optimum solution will be of cost 1.

5.2.3 SAT modulo theories (SMT)

SAT modulo theories (SMT) instances are logic formulas where some boolean variable have been replaced by atoms from some background theory like linear integer arithmetic, difference logic, arrays, etc.

For instance, in the following SMT formula, $(x \geq 2) \vee ((2+x=y) \rightarrow z = 1)$, x, y and z are integer variables inside linear integer arithmetic atoms.

5.2.4 Integer Linear Programming (ILP)

Integer Linear Programming (ILP) problems are a mathematical optimization problem in which the objective function and the constrains are lineal, and all the variables are restricted to be integers.

A variant of the Integer Lineal Programming is the Mixed Integer Lineal Programming (MILP) which involve problems where only some of the variables have to be integers, while the other variables may be non-integers.

5.3 Implied constraints

Implied constraints are constraints that can be added to a model without changing its set of solutions. It is useful to add these constraints since usually they increase the propagation capabilities of the solver.

Using implied constraints has a drawback, by using them the size of the model is increased, since we are adding redundant information. It is important to have this in mind since sometimes what is best is to find a compromise between the implied constraints used and the size by which the model is increasing.

The use of implied constrains has been proved several times in the literature to have benefits when encoding into SAT, see for instance [11] [2] [3].

5.4 Symmetries

When solving a combination problem, sometimes there are interpretations of the problem that can be obtained by applying a linear operation from another interpretation. If these interpretations are models it is not a (big) problem since we are obtaining more (symmetric) solutions for the problem.

The trouble come when these interpretations are not models, and it means that the solver is making symmetric failures that could be avoided without losing essential solutions.

Break symmetries is highly recommendable since it reduces the search tree of the problem. It is also important to notice that for breaking symmetries it is necessary to add more constraints to the model, which means to have bigger models.

In figure 3 there is an example of symmetric solutions for the famous problem of the 5-queens where the goal is to put five queens on a 5x5 chess board without them threatening each others. There we have two solutions where the second one is the first one rotated 90 degrees.

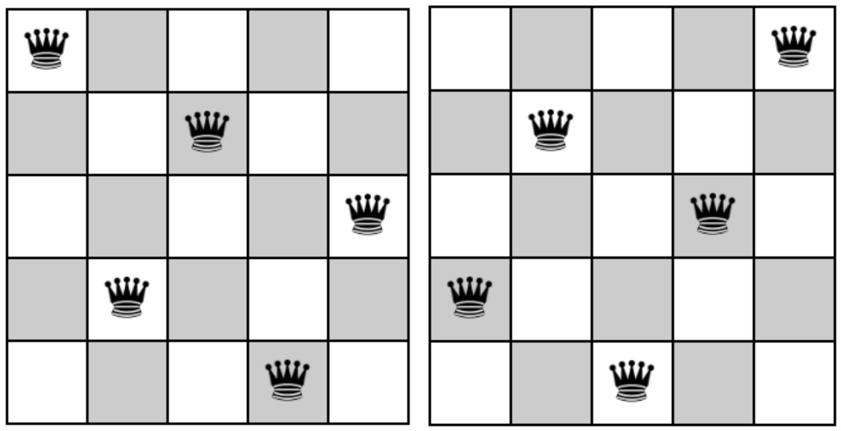

Figure 3: Two symmetric solutions for the 5-queens problem. The second solution is the first one rotated 90º.

5.5 Instances types

An instance is a representation of a problem, and these instance can be grouped in two categories depending on the way that the have been obtained. These categories are:

- Industrial instances, or real-world instances. These are the ones that are obtained from the real world, and it means that they are real instances. Usually these are the most difficult to obtain and also the ones more wanted.

- Random instances. These are the instances that are obtained randomly. These instances are easy to obtain since it is fairly easy to create a random instance generator.

Create industrial instances is a very interesting problem. For doing this, it is necessary to analyse real world instances for a specific problem and then generate instances that follow the patters that have been observed. The quality of these instances depends on the quality of the analysis of the real world instances.

This distinction is quite important since that a solver that is good solving random instances may not be that good solving real world instances, and vis-versa.

6 Requirements and decisions

In this project we have different goals, on one hand we want to compare the performance of different MaxSAT models of the B2B scheduling optimization problem. And on the other hand we want to dig into the use of implied constraints. Also, to make all of this possible it is necessary to have lots of instances, and thats why we are also building a instance generator.

In order to accomplish this objectives we had to made some decisions about how we were going to do this.

6.1 Software requirements

Our aim is to model the B2B scheduling problem as a MaxSAT problem and solve it. For doing so we have to phases:

- Transform our instances of the B2B scheduling problem into MaxSAT instances. This is done by writing a JAVA (jdk 7) application that will take as input a B2B instance and will give as a output the equivalent MaxSAT instance. This Java application is a console application since it is no necessary to have any kind of user interface. It was decided to use Java as a programming language for this phase because of the quality and quantity of the Java documentation and libraries available, and also because Java is a language that we were already familiarized with.

- Solve the MaxSAT instance. For this phase we use two different state-of-the-art MaxSAT solvers. For the comparison of the different MaxSAT models, we use QMaxSat14.04auto-glucose3 [13], which uses glucose [5] [10] as a SAT solver, and at its turn, glucose is based on MiniSat [11]. QMaxSat14.04auto-glucose3 did a very good job in the industrial partial MaxSAT tracks on the MaxSAT Evaluation 2014. [12].

  For the exhaustive analysis of the implied constraints we use Open-WBO [14][13] as MaxSAT solver, also using glucose as SAT solver. This

[10]`http://www.labri.fr/perso/lsimon/glucose/`
[11]`http://minisat.se/`
[12]`http://www.maxsat.udl.cat/14/results/index.html`
[13]`http://sat.inesc-id.pt/open-wbo/`

solver was ranqued as one of the best non-portfolio solver in the industrial partial MaxSAT tracks of the MaxSAT Evaluations 2014 and 2015.

The reason of using two different MaxSAT solvers is that between the first set of experiments and the second set of experiments there was and elapse of time in which Open-WBO seem to rise its position on the MaxSAT Evaluation. Notice that the fact of using two different MaxSAT solvers in this project is not a problem an does not led to erroneous conclusions since the experiments that use one or other MaxSAT solver are not compared between them.

6.2 Instances

Because of the nature of our project, it is highly necessary to have a big set of instances to run our models with. We already had nine instances (four industrial instances, and five crafted instances) that came from the work in [6]. For this project we also manually craft eleven new instances which are used to compare the new models with the existent ones.

However, it is not enough for the third goal of the project (to make an exhaustive study of the use of implied constraints). For this reason we model an instance generator, doing so we can have lots of instances that share some of the real-wold main characteristics. This instance generator is written in *C++* for the speed that offers this language.

6.3 Hardware requirements

In order to run our experiments we used a cluster with 13 machines, each machine with the following specifications:

All experiments have been run on a cluster of Intel Xeon$^{\text{TM}}$CPU@3.1GHz machines, with 8GB of RAM, under 64-bit CentOS release 6.3, kernel 2.6.32.

7 Definition of the problem

Since all this project and work started when the *"Parc científic i tecnologic de la Universitat de Girona"* came asking for a solution to automatize the timetable generation for the event *"Forum del parc científic i tecnologic de la Universitat de Girona"*, we will start here explaining what is the process in that event, which will be similar to any other B2B event. And then we will formalize the problem.

7.1 B2B events

As said before B2B are events where brief meetings between participants with similar interests are celebrated. In the *"Forum del parc científic i tecnologic de la Universitat de Girona"* the participants filled in a form talking about their interests. Then the participants can see the information of every other participant, and they request meetings with normal or hight priority. The participants can also add some other information like their available time or if they prefer to have meetings only on the morning or the afternoon.

After having gathered all this information, we do a prune of the meetings according to its priority.

Thus the information that we have at this point is:

- The requested meetings, after being filtered, for each participant.
- The forbidden time slots of each participant.
- Information about the event, like how many meetings can be hold at the same time, duration of each meeting, etc.

Now we can start with the problem itself, we have to find a schedule that fulfils the following restrictions:

- Each meeting has to be scheduled at morning or afternoon as requested.
- Each participant has at most one meeting at each time slot.
- No meetings can be scheduled on participant's forbidden time slots.
- At most one meeting is scheduled in a given time slot and location.

Once the final schedule is obtained, the human expert asks the participants for a confirmation. The confirmed ones are fixed and the rejected ones are retracted. Finally the last arrival participants have to be considered, they may ask for some new meetings. Add this point the human expert tries to add manually the last-minute meetings.

In the figure 4 a diagram of this process is shown.

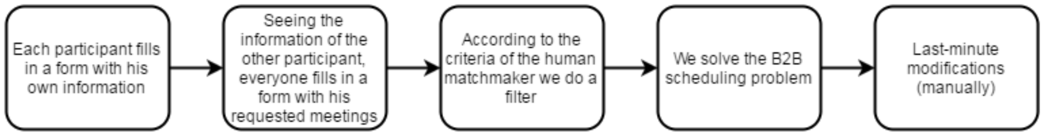
Each participant fills in a form with his own information
Seeing the information of the other participant, everyone fills in a form with his requested meetings
According to the criteria of the human matchmaker we do a filter
We solve the B2B scheduling problem
Last-minute modifications (manually)

Figure 4: Diagram explaining the process of a B2B event

7.2 B2B Scheduling Problem (B2BSP)

Here we formally define the B2B Scheduling Problem. Let us define some nomenclature first.

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

Definition 2 (B2B Scheduling Problem)**.** The B2B Scheduling Problem (B2BSP) is the problem of finding a feasible B2B schedule.

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

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

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

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

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

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

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

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

$$\forall m \in M : \\ (t(m) = 1 \Longrightarrow \pi_1(S(m)) \in T_1) \;\wedge\; (t(m) = 2 \Longrightarrow \pi_1(S(m)) \in T_2)$$

7.3 B2B Scheduling Optimization Problem with homogeneity d (B2BSOP-d)

Typically, we are interested in schedules that minimize the number of idle time periods. By an *idle time period* we refer to a group of idle time slots between a meeting of a participant and her next meeting.

Before formally defining this optimization version of the B2BSP, we need to introduce some auxiliary definitions.

Definition 3 (Idle time period)**.** Given a B2B schedule S for a set of meetings M, and a participant $p \in P$, we define $L_S(p)$ as the list of meetings in M involving p, ordered by its scheduled time according to S:

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

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

Definition 4 (B2B Scheduling Optimization Problem with homogeneity h)**.** The *B2B Scheduling Optimization Problem with homogeneity h (B2BSOP-h)* is the problem of finding a feasible B2B schedule S, where the total number of idle time periods of the participants is minimal, i.e., minimizes

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

and also the difference between the number of idle time periods of any two participants is at most h.

7.4 NP-Hardness of B2BSP

The complexity hardness of the B2BSP can be proved with a reduction from the edge coloring problem that is NP-complete. Let us define the Edge coloring problem with more detail.

Definition 5 (Edge coloring problem)**.** Decide whether given a graph G, it is possible to color all edges of G in such a way that any two consecutive edges have different color using only $degree(G)$ colors.

In Figure 5 there are some examples of graphs with a proper edge coloring.

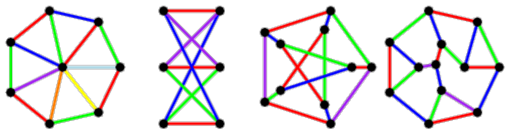

Figure 5: Four examples of a proper edge coloring.

Once we know the edge coloring problem, let us show how the B2BSP can be reduce to it:

- Each node of the graph represents a participant of the B2B event.
- Each edge between two nodes corresponds two nodes corresponds to a meeting between the corresponding participants.
- The degree of the graph is the number of time-slots.
- For the reduction we can assume that we have as many tables as we need.

Also, the B2BSP problem can be reduced from the restricted timetable problem (RTT) as proved in [9]

8 Implementation

In this section we describe our MaxSAT models for the B2BSOP-d problem and our instance generator. Firstly we start with out MaxSAT models starting with the one called *Base Model*. This *Base Model* contains all the necessary constraints to solve the problem. After we extend the model by presenting the two implied constraints that we have identified, and then we present the constraints for doing the symmetry breaking.

Finally we present our random B2B instance generator where we will give the notions of shape and density of a B2B instance and we explain how we have build it.

8.1 MaxSAT Base Model for the B2BSOP-d

8.1.1 Parameters

Each instance is defined by the following parameters.

nMeetings: number of meetings
nTimeSlots: number of available time slots
nMorningSlots: number of morning time slots
nTables: number of available locations
nParticipants: number of participants
morningMeetings: subset of $\{1, \ldots, nMeetings\}$ to be scheduled in morning slots
afternoonMeetings: subset of $\{1, \ldots, nMeetings\}$ to be scheduled in afternoon slots
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

8.1.2 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

$max_1, \ldots, max_{\lfloor (nTimeSlots-1)/2 \rfloor}$ and $min_1, \ldots, min_{\lfloor (nTimeSlots-1)/2 \rfloor}$: unary representation of an upper bound and a lower bound of the maximum and minimum number of idle time periods among all participants, respectively. Note that there can be at most $\lfloor (nTimeSlots - 1)/2 \rfloor$ idle time periods per participant. By restricting the difference between these variables to be less than a certain value, we will enforce homogeneity of solutions (Constraints (16) to (21)).

Notice that all variables but *schedule* are only necessary for optimization. We also use some auxiliary variables that will be introduced when needed.

8.1.3 Constraints

All constraints except (15) are hard. To help readability we define $M = \{1, \ldots, nMeetings\}$, $T = \{1, \ldots, nTimeSlots\}$, $P = \{1, \ldots, nParticipants\}$.

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

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

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

$$\bigwedge_{i \in meetings(p),\, j \in forbidden(p)} \neg schedule_{i,j} \qquad \forall p \in P \qquad (2)$$

- *Each meeting having a morning or afternoon slot requirement is scheduled in a time slot of the appropriate interval.*

$$exactly(1, \{schedule_{i,j} \mid j \in 1..nMorningSlots\}) \qquad \forall i \in morningMeetings \qquad (3)$$

$$\neg schedule_{i,j} \qquad \begin{array}{l} \forall i \in morningMeetings \\ \forall j \in nMorningSlots + 1..nTimeSlots \end{array} \qquad (4)$$

$$exactly(1, \{schedule_{i,j} \mid j \in nMorningSlots + 1..nTimeSlots\}) \qquad \forall i \in afternoonMeetings \qquad (5)$$

$$\neg schedule_{i,j} \qquad \begin{array}{l}\forall i \in afternoonMeetings\\ \forall j \in 1..nMorningSlots\end{array} \tag{6}$$

$$exactly(1, \{schedule_{i,j} \mid j \in T\}) \qquad \forall i \in M \setminus (morningMeetings \cup afternoonMeetings) \tag{7}$$

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

$$atMost(nTables, \{schedule_{i,j} \mid i \in M\}) \qquad \forall j \in T \tag{8}$$

Notice that with Constraints (3) to (8) we get a total mapping from the meetings to time slots and locations.

8.1.4 Optimization

Minimization of the number of idle time periods is achieved by means of soft constraints. In order to be able to perform that minimization 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 participants of the meeting.*

$$schedule_{i,j} \rightarrow (usedSlot_{p_i^1,j} \wedge usedSlot_{p_i^2,j}) \qquad \forall i \in M, j \in T$$
$$\text{where } p_i^1 \text{ and } p_i^2 \text{ are the participants of meeting } i. \tag{9}$$

  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.

$$usedSlot_{p,j} \rightarrow \bigvee_{i \in meetings(p)} schedule_{i,j} \qquad \forall p \in P, j \in T \tag{10}$$

- *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.*

$$\neg usedSlot_{p,1} \rightarrow \neg fromSlot_{p,1} \quad \forall p \in P \tag{11}$$

$$(\neg fromSlot_{p,j-1} \wedge \neg usedSlot_{p,j}) \rightarrow \neg fromSlot_{p,j} \quad \forall p \in P, j \in T \setminus \{1\} \tag{12}$$

$$usedSlot_{p,j} \rightarrow fromSlot_{p,j} \quad \forall p \in P, j \in T \tag{13}$$

$$fromSlot_{p,j-1} \rightarrow fromSlot_{p,j} \quad \forall p \in P, j \in T \setminus \{1\} \tag{14}$$

– [Soft constraints] *If some participant does not have any meeting in a certain time slot, but it has had some meeting before, then she does not have any meeting in the following time slot.*

$$(\neg usedSlot_{p,j} \wedge fromSlot_{p,j}) \rightarrow \neg usedSlot_{p,j+1} \quad \forall p \in P, j \in T \setminus \{nTimeSlots\} \tag{15}$$

We claim that, with these constraints, an optimal solution will be one having the least number of idle time periods. Note that, for each participant, each meeting following some idle time period increases the cost by 1.

If we were just considering optimization, Constraints (11) and (12) would not be necessary, since minimization of the number of idle time periods would force the value of $fromSlot_{p,j}$ to be false for every participant p and time slot j previous to the first meeting of p. However, since we are also seeking for homogeneity, these constraints are mandatory. Without them, the value of $fromSlot_{p,j}$ could be set to true for time slots j previous to the first meeting of p, inducing a fake idle time period in order to satisfy the (hard) homogeneity constraints defined below.

8.1.5 Homogeneity

We reify the violation of soft constraints in order to count the number of idle time periods of each participant. This will allow us to find the maximum and minimum number of idle time periods among all participants, and to enforce homogeneity by bounding their difference.

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

$$endHole_{p,j} \leftrightarrow \neg\big((\neg usedSlot_{p,j} \wedge fromSlot_{p,j}) \rightarrow \neg usedSlot_{p,j+1}\big) \qquad \forall p \in P, j \in T \setminus \{nTimeSlots\} \quad (16)$$

- *$sortedHole_{p,1}, \ldots, sortedHole_{p,nTimeSlots}$ are the unary representation of the number of idle time periods of each participant p.*

$$sortingNetwork([endHole_{p,j} \mid j \in T], [sortedHole_{p,j} \mid j \in T]) \qquad \forall p \in P \quad (17)$$

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

$$sortedHole_{p,j} \rightarrow max_j \qquad \forall p \in P, j \in 1..\lfloor (nTimeSlots-1)/2 \rfloor \quad (18)$$
$$\neg sortedHole_{p,j} \rightarrow \neg min_j \qquad \forall p \in P, j \in 1..\lfloor (nTimeSlots-1)/2 \rfloor \quad (19)$$

  Constraints (18) and (19) are not enough to ensure that the *max* and *min* variables exactly represent the maximum and minimum number of idle time periods among all participants. However, together with Constraints (20) and (21), they suffice to soundly enforce the required homogeneity degree.

- *The difference between the maximum and minimum number of idle time periods can be at most d (in our setting the chosen number was 2).*

$$dif_j \leftrightarrow min_j\, XOR\, max_j \qquad \forall j \in 1..\lfloor (nTimeSlots-1)/2 \rfloor \quad (20)$$
$$atMost(d, \{dif_j \mid j \in 1..\lfloor (nTimeSlots-1)/2 \rfloor\}) \quad (21)$$

8.2 Implied Constraints

We have identified the following implied constraints.

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

$$exactly(|meetings(p)|, \{usedSlot_{p,j} \mid j \in T\}) \qquad \forall p \in P \quad (22)$$

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

$$atMost(2 \times nTables, \{usedSlot_{p,j} \mid p \in P\}) \qquad \forall j \in T \qquad (23)$$

8.3 Symmetry Breaking

For the B2BSOP we consider location and time symmetries. Location symmetries are implicity eliminated by the model described, since only the number of tables occupied is considered. Unfortunately, removing time symmetries in the presence of participants' forbidden time slots seems not to be feasible. However, we can break some time symmetries when there are no forbidden time slots and the meetings have neither morning nor afternoon slot requirements (there are several instances with these characteristics). Notice that since we are minimizing the number of idle time periods, we cannot soundly break time symmetries by simply fixing a priori an ordering of meetings. Instead, what we do is to force some ordering in the "matrix" of *usedSlot* variables as follows, assuming an even number of time-slots and the existence of a participant with an odd number of meetings.[14]

- *For some participant p with an odd number of meetings we force the number of meetings of p taking place in the first half of time slots to be odd.*

$$((\ldots(usedSlot_{p,1}\ XOR\ usedSlot_{p,2})\ \ldots\ )\ XOR\ usedSlot_{p,\lfloor nTimeSlots/2 \rfloor}) \qquad (24)$$

8.4 Encoding of Global Constraints

In the model described appear the global constraints at most (*atMost*) and exactly (*exactly*) k. These global constraints state that at most or exactly k variables of a given set of variables needs to be *true* respectively. In the model also appears the global constraint sort *sortingNetwork*, that given a set of variables as a input it returns as a output the same number of variables, first the ones that are *true*, and then the ones that are $false$.

Here we first recall the encodings that were used for these global constraints in the MaxSAT model presented on[15], from now on this will be called *Naïve Encoding*. Then we state the encodings that we use for the *Cardinality Networks based Encoding* that we present in this project.

[14]All instances considered are like this.

Naïve Encoding

- $atMost(1, \_)$: quadratic number of pairwise mutex clauses.
- $exactly(1, \_)$: $atMost(1, \_)$ plus a clause (disjunction) with all the involved variables, for the "*at least* part of the constraint.
- *sortingNetwork*: odd-even sorting network.
- $atMost(k, \_)$: naïve sequential unary counter [18].

Cardinality Nerworks based Encoding

- $atMost(1, \_)$: quadratic number of pairwise mutex clauses.
- $exactly(1, \_)$: commander-variable encoding [12].
- $exactly(k, \_)$, *sortingNetwork* and $atMost(k, \_)$: cardinality networks [1] [4].

By using cardinality networks we can deal with soft constraints in a more clever way: instead of soft constraints (15), we post as soft constraint the negation of each "output variable" $sortedHole_{p,j}$ of the *sortingNetwork* corresponding to constraint (17). This way, knowing that each participant will have at most $\lfloor (nTimeSlots - 1)/2 \rfloor$ idle time periods, we can reduce the number of soft constraints, as well as the number of $sortedHole_{p,j}$ variables of each participant, to a half.

8.5 Random B2B Instance Generator

Here we present our random B2B instances generator. To create this generator, we consider a number of participants P and a number of meetings 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.

We first analyze how these distributions of meetings among participants are in at hand real-world B2B instances. The set of available real-world B2B instances [6] plus the eleven new instances that we provide [15] sum up a total

[15]Instances available at `http://imae.udg.edu/recerca/lap/simply/`

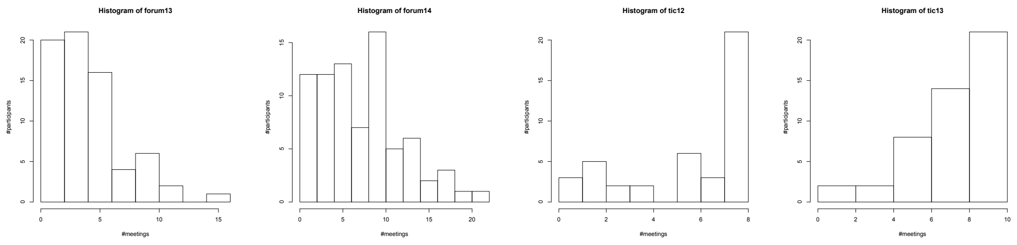

Figure 6: Histograms of variable: number of meetings of each participant, for the four real-world instances.

of 20 instances: 5 of them come from real data, and the remaining 15 are crafted modifications from the real ones. We focus our analysis on 4 real B2B instances (the fifth one is similar to the two `tic` instances considered).

In a first step we want to check if our real data fits any known distribution. As we analyze the number of meetings requested by each participant, we do not consider if a meeting is requested by both of its two participants or just by one. Also, our analysis does not consider others aspects of the distribution of requests. For instance, if requests are *clustered*. This may happen in B2B events divided into fields, in which participants request most of their meetings with other participants of the same field, creating *clusters* of participants.

In Figure 6 we represent the histograms of the number of meetings of each participant in real-world B2B instances. We observe that some of the histograms are right-skewed (`forum-13` and `forum-14`) while some others are clearly left-skewed (`tic-12` and `tic-13`). From the shape of those histograms, it is difficult to conjecture the probability distribution for such data. Right-skewed histograms slightly resemble those of a binomial distribution $B(n, p)$ with small n and small p. For the left-skewed histograms, it is worth remarking that the number of available time slots T for meetings was unknown before the requesting process. For that reason, some requests were removed (by a priority criteria) of those participants having more requests than T. This preprocessing step was performed in all instances in order to ensure the feasibility restriction mentioned above.

As we cannot draw general conclusions about probability distributions from this reduced set of real instances, we present a regular model to generate random B2B instances. This model is based on a probability U that a participant requests a meeting with another.

8.5.1 B2B random generator regular model

Definition 6 (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 $i \neq j$.

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

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 $p = U$.

Therefor in the model presented, the expected number of meetings requested by each participant is $(P - 1)U$. since 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.[16] This way, the resulting problem contains many participants requesting T meetings, i.e., many observations close to the maximum. This is not a problem since we have found this kind of participant is very frequent.

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 checks whether or not it is beneficial to use them for the B2BSOP, 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*.

Let us define first the assignation matrix A of size $T \cdot L$. An element

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

(i, j) of this matrix represents that time slot t_i and location l_j is scheduled whether with a meeting $m_k \in \mathcal{M}$ or with no meeting.

$$A_{i,j} = \begin{cases} m_k & \text{if } 1 \leq k \leq M \\ \varnothing & \text{otherwise} \end{cases}$$

Definition 7 (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 $T \cdot L$.

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

Definition 8 (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.

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

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. The model also does not take into account forbidden time slots for the participants nor morning/afternoon requirements. However, our model seems adequate to model B2B instances similar to the known real-world ones.

9 Results

Here we present the results that we have obtained. We start with the results related with the first goal of the project, they are exposed in the subsection *Comparison with previous work* 9.1. We follow with the results of the second and the third goal, exposed in the subsection *Implied constraints experimental evaluation* 9.2.

Recall that the second goal was a mean to accomplish the last goal, so it make sense to present the results referring to them both in the same subsection.

9.1 Comparison with previous work

In this section we compare the performance of the MaxSAT model proposed in [15], which is the one called *Naïve*, with the different models proposed here. We also show how extending the model with implied constraints and symmetry breaking, we can significantly improve the solving time. We use the same nine instances that the authors used in [6], plus new eleven instances. Among all these instances, there are five of them that come from the real world (the ones without *craf* annotation) the rest have been crafted from those by increasing the number of meetings, reducing the number of locations and removing the forbidden time slots.

All the MaxSAT instances are solved by the state-of-the-art MaxSAT solver *QMaxSat14.04auto-g3* [13].

All experiments have been run using the default options of each solver, on Intel® Xeon™CPU@3.1GHz machines, under CentOS release 6.3, kernel 2.6.32.

In Table 1 we show the results obtained. Only instances named *tic* do not contain forbidden time slots nor morning and afternoon preferences, hence symmetry related experiments are only reported for those. Column named **naïve** shows the results obtained using the *Naïve Encoding*. Columns **cardinal** show the results using the *Cardinality Network Based Encoding* of our base model. Columns **imp1**, **imp2**, **imp12** and **imp12+sym** show the results for the *Cardinality Network Based Encoding* using implied constraints 1, 2, both, and both with symmetry breaking, respectively. The three numbers below the names of each instance are: the ratio between the median of meetings per participant and *nTimeSlots*, the ratio between *nTables* and *nParticipants*, and the ratio between the number of meetings to schedule and the available slots ($nTables \times nTimeSlots$).

instance	naïve		cardinal		imp1		imp2		imp12		imp12+sym	
forum-13 (0.20, 0.40, 0.52)	24.8	0	20.5	0	13.4	0	25.2	0	18.3	0	-	-
forum-13crafb (0.24, 0.36, 0.66)	1492.7	6	83.4	6	82.4	6	83.1	6	81.2	6	-	-
forum-13crafc (0.20, 0.34, 0.61)	116.3	1	1872.4	1	1661.3	1	1800.5	1	1300.2	1	-	-
forum-14 (0.35, 0.56, 0.62)	TO	-	431.2	2	349.1	2	409.2	2	240.2	2	-	-
forumt-14 (0.79, 0.90, 0.87)	21.1	5	8.0	5	8.5	5	11.9	5	10.2	5	-	-
forumt-14crafc (0.79, 0.83, 0.94)	148.9	5	32.7	5	28.8	5	33.1	5	31.5	5	-	-
forumt-14crafd (0.78, 0.83, 0.94)	84.9	4	32.4	4	26.6	4	37.1	4	35.5	4	-	-
forumt-14crafe (0.78, 0.80, 0.98)	TO	-	95.2	5	78.1	5	105.2	5	94.7	5	-	-
ticf-13crafa (0.21, 0.40, 0.52)	21.2	0	24.6	0	15.0	0	45.9	0	35.9	0	-	-
ticf-13crafb (0.51, 0.36, 0.66)	3866.1	3	118.3	3	117.3	3	111.3	3	114.2	3	-	-
ticf-13crafc (0.21, 0.34, 0.61)	309.4	1	574.2	1	562.3	1	416.9	1	432.3	1	-	-
ticf-14crafa (0.35, 0.56, 0.62)	TO	-	TO	-	1532.8	0	2044.1	0	1339.6	0	-	-
tic-12 (0.74, 1.00, 0.74)	0.2	0	0.2	0	0.3	0	0.2	0	0.2	0	0.4	0
tic-12crafc (0.74, 0.76, 0.97)	7.8	0	4.1	0	3.1	0	2.5	0	2.6	0	3.4	0
tic-13 (0.76, 0.89, 0.85)	18.4	0	5.9	0	4.1	0	4.6	0	4.2	0	5.7	0
tic-13crafb (0.80, 0.89, 0.87)	3.6	0	2.4	0	2.6	0	7.1	0	5.5	0	4.1	0
tic-13crafc (0.76, 0.80, 0.94)	TO	4	25.9	4	19.1	4	25.2	4	23.9	4	26.1	4
tic-14crafa (0.79, 0.90, 0.87)	30.0	0	16.3	0	10.2	0	24.4	0	16.4	0	14.2	0
tic-14crafc (0.79, 0.83, 0.94)	740.0	0	49.3	0	45.1	0	45.7	0	44.5	0	56.8	0
tic-14crafd (0.79, 0.83, 0.94)	190.7	0	35.2	0	47.9	0	32.5	0	34.9	0	53.2	0

Table 1: Solving time (in seconds) and optimum found (number of idle time periods) per instance and solver. TO stands for 2 hours timeout. For aborted executions we report the (sub)optimum found if the solver reported any.

From the results of Table 1 we can extract the following conclusions:

- The *Cardinality Network Based Encoding* of our base model clearly outperforms the *Naïve encoding.*
- To use implied constraints is, in almost all the situations, beneficial.

- When the amount of information provided by the implied constraints is elevated is when really pays off to use them: in particular, for implied constraint 1, this happens when the ratio between the median of meetings per participant and *nTimeSlots* is low; for implied constraint 2, this happens when the ratio between *nTables* and *nParticipants* is low. This result is studied more deeply in the next subsection of results.

- The use of symmetry breaking seems not to really help (in fact we think that we need some more hard instances to appreciate its possible benefits).

9.2 Implied constraints experimental evaluation

Here, we expose the results related with the second and the third goal of this project by evaluating the effects on the solver performance by the use of implied constraints. To do so we follow the following process:

1. Generate some families of random B2B instances with the generator that we described in the previous section.

2. Solve the B2B instances with and without using implied constraints. Doing these experiments we can identify those cases for which the use of some implied constraint is beneficial.

3. Validate these observations of random B2B instances in real-world problems since it is very important to check if the behaviour observed with the random instances also happens with the industrial instances.

4. Conjecture the reasons of the success of using implied constraints based on some observations from the solver.

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 [6], we use a value for homogeneity $h = 2$. We solve each instance without using implied constraints, and with using implied constraint 1, 2 and both. In the plots, these methods are named as *no-imp*, *imp1*, *imp2* and *imp12* respectively. We use Open-WBO [14] as MaxSAT solver.

Note that the solver used here is different that the one that we use for the comparison results 9.1. The reason is because we used the state-of-the-art solver at the moment of doing the experiments, and there is an elapse of time of several months between the two experiments.

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.

9.2.1 Random B2B Instances

In our experimentation, we use an estimation of the number of meetings M, since this number is unknown *a priori*. As explained on the previous section, 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 $E[M] \approx E[R]$.[17] Finally, we approximate M using its expected value, i.e., $M \approx E[M]$. In what follows, we use these approximations.

First, we analyze our regular model varying the probability U for a fixed density and a fixed shape. Using the approximation of $M \approx E[R]$, we use a density $d \approx 1$ and a shape $s = 1$ (i.e., $T = L = \left\lceil \sqrt{E[R]} \right\rceil$). In Figure 7, we represent the runtime of solving these families of random instances, with $U = \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5\}$ and fixed $P = 40$. We observe that, in general, using no implied constraints is slower than using any of

[17]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 $E[M] \approx P(P-1)U$.

them, and using both is always one of the fastest methods. Also, in general we see that the hardness of solving these instances increases for higher values of U. Although this does not happen in all cases (e.g., the hardest family using *imp1* is observed when $U = 0.2$), that is specially clear when using both implied constraints (i.e., *imp12*), which is indeed the fastest method in general.

It is worth mentioning that when $U = 0.5$, we are solving instances with a number of meetings close to 540. These instances have a much bigger number of meetings than real-world problems, whose maximum is the 302 meetings of the instance `forum-14`. Also notice that in this case, we cannot approximate $M \not\approx P(P-1)U = 780$. For the experiments that follow, we continue using a fixed number of participants $P = 40$, which we consider that is a big enough number to reproduce the hardness of some real-world problems. Notice that the number of participants for the real instances categorized as *tic* ranges between 42 and 47. 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 next thing we want to do is detect those cases in which the use of some implied constraint is more beneficial than any other encoding. To do so, we generate families of random B2B instances varying the density and the shape.

In our results, we use the Penalized Average Runtime 10 (PAR10) representation, 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 8, 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:

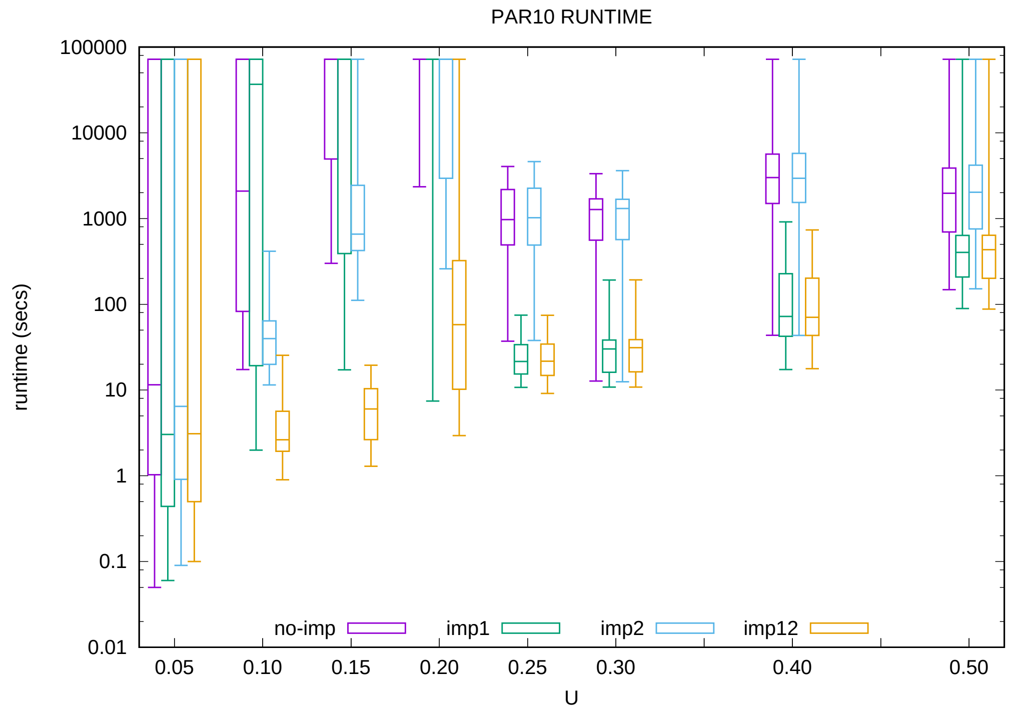

Figure 7: Analysis of the runtime (in seconds) of solving some random B2B instances with and without using implied constraints, using the regular model with $P = 40$ and $U = \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5\}$, with a fixed density $d \approx 1$ and shape $s = 1$ (i.e., $T = L = \left\lceil \sqrt{E[M]} \right\rceil$). For visualization, we slightly shift the value of U of each family.

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

Notice that when P, U and s are fixed, the variations in the density are equivalent to variations in T (or L). In this experiment we identify several phenomena:

- 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.

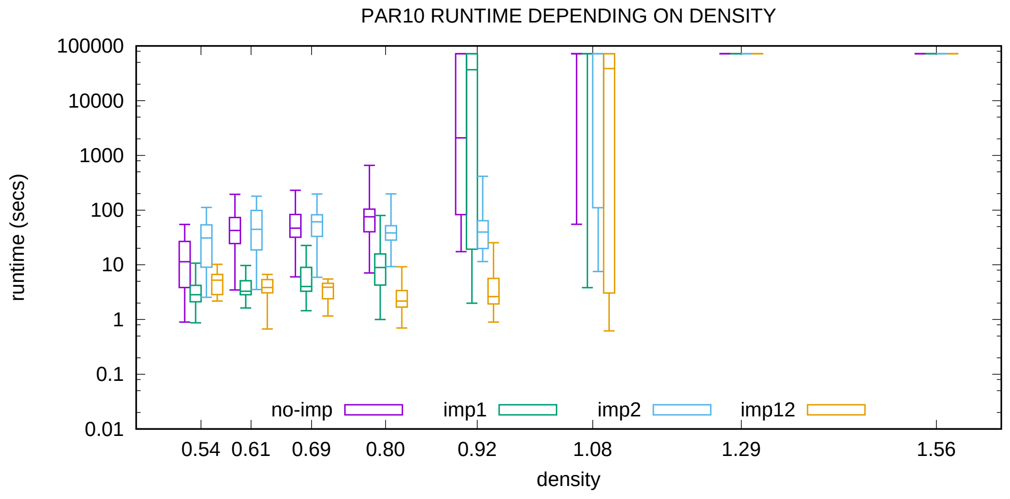

Figure 8: PAR10 (in seconds) of solving some random B2B families of instances, with and without using implied constraints, varying their density d. Instances are generated using the regular model with $P = 40$, $U = 0.1$, and a fixed shape $s = 1$ (i.e., $T = L$), hence $d = M/TL \approx P(P-1)Us/T^2$.

- 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*).

- 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 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).

In Figure 9, 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 a fixed 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

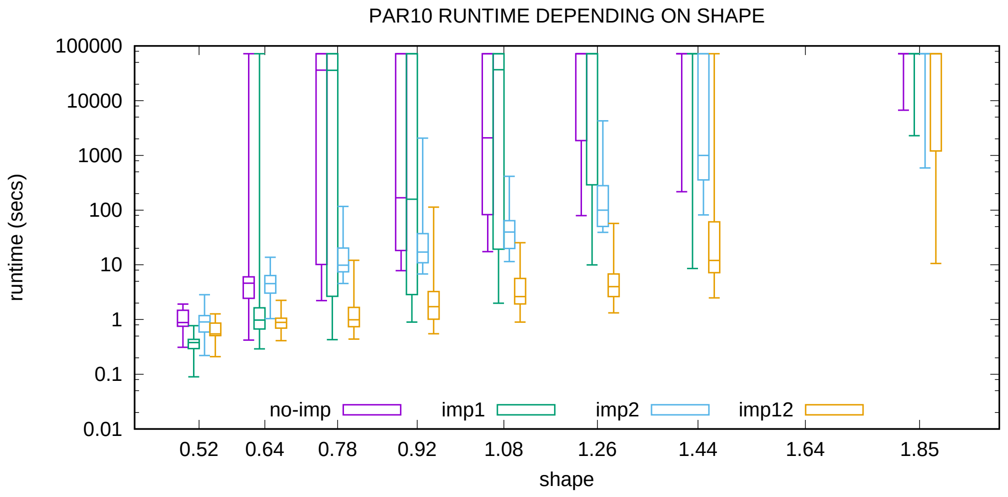

Figure 9: PAR10 (in seconds) of solving some random B2B families of instances, with and without using implied constraints, varying their shape s. Instances are generated using the regular model with $P = 40$, $U = 0.1$, and a fixed density $d \approx 1$ (i.e., $T \cdot L = E[R]$), hence $s = T/L \approx T^2/P(P-1)U$.

be *similar*[18] 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:

- 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*. Therefore, it is more likely to find a solution with no idle time periods, and the solver does not need to continue the search to prove that it is the optimum. On the contrary, when the shape is *high* (and hence, the number of time slots is *high* and the number of locations is *small*), the optimum may contain idle time periods, and proving it can be costly.

- Again *no-imp* is worse than *imp1* and *imp2*, and these two are worse

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

than *imp12*.

- The benefits of using one of the implied constraints depends 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%).

- The best encoding is the *imp12*. 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.

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.

9.2.2 Real-world B2B Instances

Next 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 10 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 the observations from the Figure 8, 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 conclude that this observation is also valid in

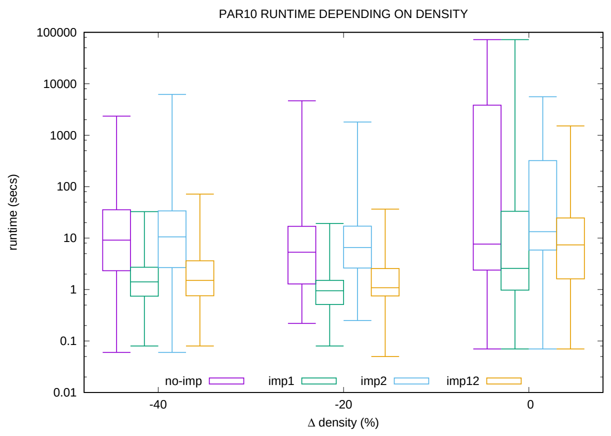

Figure 10: PAR10 (in seconds) of solving some real-world B2B instances, with and without using implied constraints, varying their density d, and with a fixed shape, i.e., $\Delta T = \Delta L$.

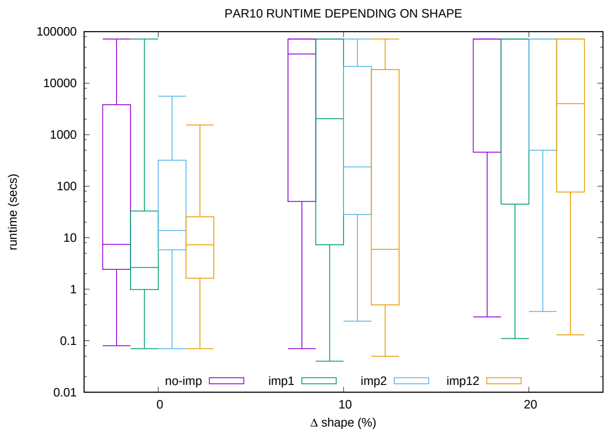

Figure 11: PAR10 (in seconds) of solving some real-world B2B instances, with and without using implied constraints, varying their shape s, and with a fixed density, i.e., fixed $T \cdot L$.

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 general, one of the best choices. This last claim is not clear from the plot. In Table 2, we report some statistics of this experiment. We use the Penalized Average Runtimes PAR1 and PAR10. Notice that the

	$\Delta d = -40\%$			$\Delta d = -20\%$			$\Delta d = 0\%$		
	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 2: Statistics of solving some real-world B2B instances, varying their density d, for a fixed shape s (with $\Delta T = \Delta L$).

	$\Delta s = 0\%$			$\Delta s = 10\%$			$\Delta s = 20\%$		
	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 3: Statistics of solving some real-world B2B instances, varying the shape s, for a fixed density d (with $T \cdot L$ fixed).

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 11 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. The observations from Figure 9 are 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 3, 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 than *imp12* dominates the other encodings. Therefore, this second observation also seems to be valid in our set of instances.

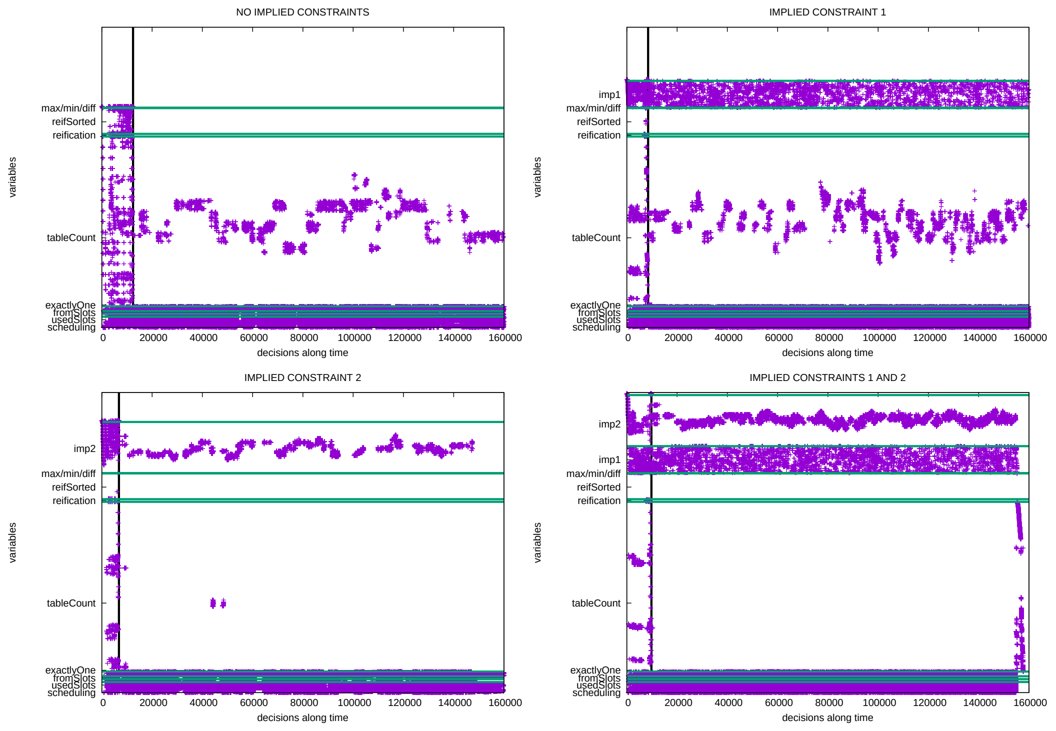

Figure 12: Branching variables decided by the solver along its execution, for the encodings *no-imp*, *imp1*, *imp2* and *imp12*, for a random B2B instance with low density, during their first 160000 decisions (solved by *imp12* in 157729 decisions.).

9.2.3 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.

In Figure 12, 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 the results from Figure 8, instances with low density are solved faster by *imp1* (and *imp12*). The runtime and the number of decisions for 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.

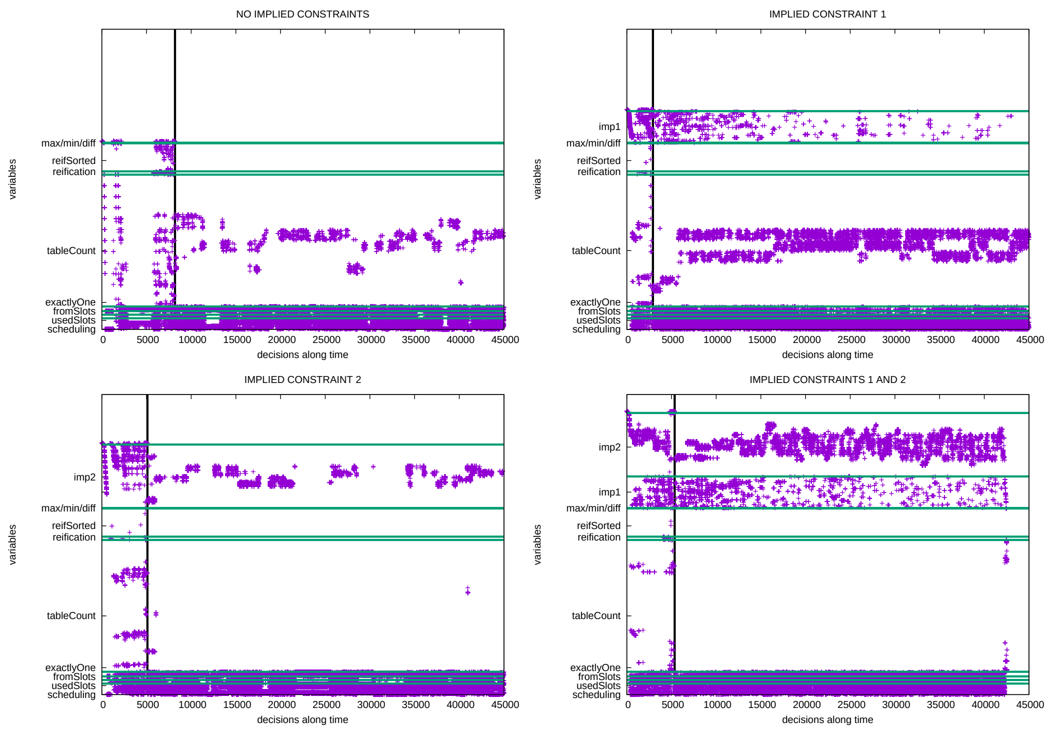

Figure 13: Branching variables decided by the solver along its execution, for the encodings *no-imp*, *imp1*, *imp2* and *imp12*, for a random B2B instance with high density, during their first 45000 decisions (solved by *imp12* in 42550 dec.).

In Figure 13, 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 the results from Figure 8, 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.

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.

10 Conclusion

In this work we have deal with the already known B2B Scheduling Optimization Problem. This is a problem that came from the real world. In this project we have deeply studied the problem, taking as a base work the one presented in [15]. At the beginning of the project we set three primordial goals.

The first goal was to improve the B2BSOP MaxSAT model. For that we look into the state-of-the-art encodings for some global constraints that we had on the MaxSAT model. Implementing these encodings, instead of the naïve ones that were used before clearly improved the time solving of the model. Moreover, we also extended the MaxSAT model in two other aspects. On the one hand, we were able to find two implied constraints which we added on the model. These two implied constraints yield really interesting results. It is important to mention that these results are the base and the motivation of the goal three. On the other hand, we extended the MaxSAT model by adding a constraint to break symmetries. This symmetry breaking can only be used for instances that do not have forbidden time slots for any participant, have a even number of timeslots, and have a participant with an even number of meetings. All of these prerequisites for the instances, and the fact that it did not seem to bring good results made our work trying to break symmetries a dead end.

We also brought eleven new instances which, together with the existent ones sum up a total of 20 real-world and crafted instances for the B2BSP.

With all of that we presented a new B2BSOP MaxSAT model (*imp12*) that has been able to solve all 20 instance (in a 2 hours timeout) of the problem in 3866 seconds while the old model could solve only 16 instances (in a 2 hours timeout) and took 7076.1 second only for these.

Respect the second goal, which was to create a B2BSP instance generator, we conclude that there are too few real-wold instances of the B2BSP. However, we proposed a random B2B instance generator which is based on a probability that a participant requests a meeting with another. This model, even thought it proved successful for our needs, does not represent some features of real-world instances, e.g., participants that do not request any meeting, forbidden time slots, morning/afternoon requirements.

Finally, related with the last goal, we provided an experimental study of the effectiveness of using implied constraints in B2B scheduling problems using MaxSAT-based encodings. For that, using our generator, we generated families of random B2B instances, and we studied the strengths and weakness of using implied constraints based on the characteristics of the instance. We focused 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 some kind of duality in the benefits of the use of the two implied constraints. 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 conjectured why this is the case by some observation of the solver. We also illustrated how the solver focuses its search when both implied are used.

11 Further work

As future work, we plan to find some more implied constraints and to improve symmetry breaking. We also plan to develop a portfolio with all the encodings and models presented. We also 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.

Referring to the B2B instance generator, we plan to do a deeper analysis of the real-world instance and take into account some of they features and characteristics, e.g., participants that do not request any meeting, forbidden time slots, moorning/afternoon requirements.

Finally, we also think in doing some modifications on the B2BSOP in order to make it more generic, e.g., establish some kind of precedence between meetings so the participants can decide which meetings they want to have firsts, this has a great interest since it is possible that some participant wants to meet another participant only hafter having had a meeting with someone else. We also consider the possibility of having meetings with more than two participants, and also in changing the optimization function so the length of the idle time periods is taken into account.

12 Related publications

Based on this work, we have published two papers:

- M. Bofill, M. Garcia, J. Suy, and M. 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.
- M. Bofill, M. Garcia, J. Giraldez-Cru, and M. Villaret. A Study on Implied Constraints in a MaxSAT Approach to B2B Problems. In *Pragmatics of SAT*, 2016.

References

[1] I. Abìo, R. Nieuwenhuis, A. Oliveras, and E. Rodrìguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 80–96. Springer, 2013.

[2] T. Alsinet, R. Béjar, A. Cabiscol, C. Fernández, and F. 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] C. Ansótegui, A. del Val, I. Dotú, C. Fernández, and F. Manyà. Modeling choices in quasigroup completion: SAT vs. CSP. In *19th National Conference on Artificial Intelligence, AAAI 2004*, pages 137–142, 2004.

[4] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[5] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solvers. In *IJCAI*, volume 9, pages 399–404, 2009.

[6] M. Bofill, J. Espasa, M. Garcia, M. Palahí, J. Suy, and M. 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.

[7] M. Bofill, M. Garcia, J. Giráldez-Cru, and M. Villaret. A Study on Implied Constraints in a MaxSAT Approach to B2B Problems. In *Pragmatics of SAT*, 2016.

[8] M. Bofill, M. Garcia, J. Suy, and M. 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.

[9] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193. IEEE, 1975.

[10] M. Gebser, T. Glase, O. Sabuncu, and T. Schaub. Matchmaking with answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 342–347. Springer, 2013.

[11] H. A. Kautz, Y. Ruan, D. Achlioptas, C. P. Gomes, B. Selman, and M. E. Stickel. Balance and filtering in structured satisfiable problems. In *17th International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 351–358, 2001.

[12] W. Klieber and G. Kwon. Efficient CNF encoding for selecting 1 from N objects. In *Fourth Workshop on Constraints in Formal Verification, CFV*, 2007.

[13] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A Partial Max-SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

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

[15] M. G. Oliveras. Scheduling b2b meetings. Projecte de Final de Grau in Computer Enginyering presented in University of Girona (UdG), 2014.

[16] G. Pesant, G. Rix, and L. 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.

[17] B. Selman, H. A. Kautz, and D. 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.

[18] C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *11th International Conference on Principles and Practice of Constraint Programming, CP 2005*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.