Downloaded from orbit.dtu.dk on: Mar 27, 2025

DTU Library

Data Reductions for the International Timetabling Competition 2019 Problem

Holm, Dennis Søren

Link to article, DOI:
10.11581/DTU.00000241

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Holm, D. S. (2022). *Data Reductions for the International Timetabling Competition 2019 Problem*. Technical University of Denmark. https://doi.org/10.11581/DTU.00000241

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Data Reductions for the International Timetabling Competition 2019 Problem

Dennis S. Holm[1,*]

Technical University of Denmark, DTU Management, Akademivej, Building 358, 2800 Kgs. Lyngby, Denmark

Abstract

When solving real-world optimization problems with large datasets, we often encounter redundancies. The data may originate from different systems or employees, leading to a loss of overview. Some structures may not individually cause redundancy, but the combination will. It is also the case for the International Timetabling Competition 2019 (ITC 2019) problem. The ITC 2019 considers a generalized formulation of the university course timetabling problem, where data are collected from 10 institutions on five continents. An example of redundancy is that a class's available times contain infeasible entries because constraints disallow the time assignment. The employee entering the valid class times may not know the limitations or have the necessary overview of the whole dataset to make such a derivation. This paper shows an algorithm to reduce the available times and rooms for classes in the ITC 2019 data. The paper introduces methods from graph theory to efficiently reduce the number of decision variables. These methods are applicable for other problems formulated as Binary/Integer Programs. Furthermore, specifically for the ITC 2019 problem, the algorithm removes redundant distribution constraints. The reduced ITC 2019 datasets are available for download at `https://doi.org/10.11583/DTU.20014070`.

Keywords: Integer Programming, Conflict graph, Data reduction, International Timetabling Competition 2019, ITC 2019

1. Introduction

It is a long-known fact that the formulation of a mathematical programming model impacts solvability. The standard solving scheme for Mixed Integer Program (MIP) includes solving the linear relaxation of the MIP model, where the integrality constraints are relaxed. Then follows a branching strategy, where variables are fixed one at a time, and the underlying model is solved until the solution has been proved optimal [13]. The linear relaxations provide a bound on the solution, which is used to evaluate the solution. The linear relaxation solution depends on the model formulation, which means that a tighter formulation leads to stronger bounds and thus fewer iterations in the branching scheme.

It has long been discussed how to reformulate the 'user-supplied' model to an improved model that is easier to solve. Various papers considering a binary program use logical tests to examine if fixing a binary variable to 0 or 1 would make the problem infeasible, deduct valid inequalities from relations in conflict graphs, tighten constraints, and remove redundancies [7, 12, 4, 8]. Later, others introduced the methods for Integer Programs (IP) and Mixed Integer Programs (MIP) as well [16, 18, 3, 5].

Some reduction methods relate to the conflict analysis of satisfiability problems (SAT). The conflict analysis for SAT problems can also be used to reduce the number of decision variables in MIP models [1]. The methods often include probing strategies using unit clauses and cliques. These methods are not only applicable to a preprocessing stage. For each infeasible branching problem, the methods can be used to deduct valid inequalities (cuts), which leads to a faster search through the branch and bound tree [1].

This paper takes the preprocessing one step back and examines the raw data for redundancies. The early examination of the problem data leads to a reduced solution space and a reduced amount of redundant constraints. The presented preprocessing techniques rely only on the raw data, making them valuable for all solution strategies, not only MIP models. Modern MIP solvers already have preprocessing procedures (known as presolve, see [2]) which are used on the 'user-supplied' MIP model. Reducing the problem instances before modeling and presolving makes the MIP model generation faster and gives a better starting point for the presolve methods.

Some optimization problems might contain features known to the user, which the presolve procedure must deduct. We show how such features can reduce the number of decision variables before creating the MIP model. Specifically, we consider a problem where different (non-overlapping) subsets of binary variables are constrained

*Corresponding author

[1]Dennis S. Holm's research project is part of the Data Science for University Management project (dsumsoftware.com) funded by MaCom A/S and Innovation Fund Denmark. Both have supported this work financially and have not participated in any research-related activities.

by set equality constraints. We introduce a conflict graph representing the relation between variable pairs where either one or the other may be equal to one. We use cliques and a probing strategy to reduce the number of variables in the model to be generated. Specifically, we introduce the reduction methods for the University Course Timetabling problem of the International Timetabling Competition 2019 (ITC 2019)[14]. Holm et al. [11] presented an algorithm for reducing the ITC 2019 problem instances using a maximal clique cover generated by a complex algorithm by Gramm et al. [6]. We show a faster and more effective reduction algorithm compared to the one by Holm et al. [11].

The paper is structured as follows. Section 2 introduces the conflict graphs and the clique reduction methods. Section 3 gives an overview the ITC 2019 problem and related conflict graphs. Section 4 and 5 shows the clique reduction and SAT reduction (probing) methods. Section 6 presents a reduction methods for redundant constraints of the ITC 2019 and Section 7 gives the complete ITC 2019 reduction algorithm. Section 8 shows the results, Section 9 states the data availability, and Section 10 concludes.

2. Conflict graphs

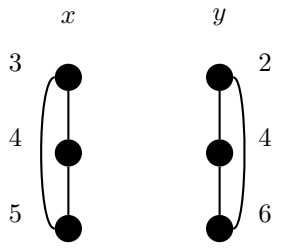
A *conflict graph* is a representation of the logical relation between the binary variables of a mathematical problem. Vertices represent the binary variables in the conflict graph, and edges represent the relation. A feasible solution cannot have both variables set to one where an edge is present. The edges can stem directly from the binary constraints $x_i + x_j \leq 1$. But can also be derived from other constraints like $3x_i + 2x_j + 2x_k + x_l \leq 4$, which would generate the edges (x_i, x_j) and (x_i, x_k).

We can use the conflict graph to find valid inequalities for the mathematical programming model. Consider a conflict graph of an optimization problem. A solution to such a problem will be represented in a graph as a subset of vertices, where no two are adjacent (an *independent set*). Any subset of vertices where two vertices are adjacent cannot represent a feasible solution since the edge symbolizes a violated constraint. However, it does not follow that any independent set is a feasible solution as other constraints not modeled in the conflict graph may exist. We look for structures such as cliques or odd holes in the conflict graph to find valid inequalities. A *clique* is a subset of vertices where each pair of vertices are adjacent. A *maximal clique* is a clique that can not be extended by adding another vertex. A *clique constraint* then represents a set of vertices where at most one can be in a feasible solution, also known as a set packing constraint, $\sum X \leq 1$. If the clique is maximal, then the clique constraint is facet defining [15]. When solving the mathematical programming model, the maximal cliques can thus be used as clique cuts.

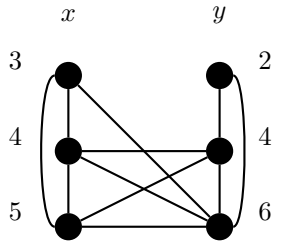
2.1. Clique reduction

Many mathematical programming problems also include set equality constraints which ensure that exactly one vertex of a subset is used. Examples include that all customers must be visited once or all jobs must be scheduled once. If a set equality constraint exists, we can use it to reduce the size of the conflict graph. A set equality constraint forms a clique in the conflict graph, which may be used to eliminate vertices of the graph. If the set equality clique is not maximal, all neighbors of the clique can be removed. If the problem contains set covering constraints, $\sum X \geq 1$, where X also forms a clique in the conflict graph, it translates to a set equality constraint $\sum X = 1$, which we may use for reducing the conflict graph.

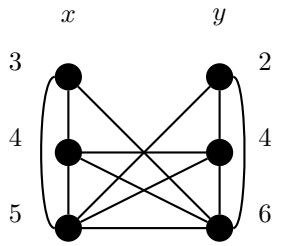
Since the conflict graphs are related to binary problems, it is not trivial that they are also applicable to Integer Programs (IP). An IP can be redefined as a binary program by enumerating all possible values of the integer variables. Consider an IP with two integer variables x and y. Let $3 \leq x \leq 5$ and $1 \leq y \leq 6$ and y must also be even and add the constraints $x + \frac{1}{2}y \leq 8$ and $|x - y| \leq 2$. First, enumerate all possible values of the variables and let them be the vertices of the conflict graph. Since x must obtain one of the integer values $\{3, 4, 5\}$, we have a clique where one vertex must be in a feasible solution. Notice that all the integer variables form such cliques, and all vertices adjacent to such a clique are removable. Figure 1 illustrates this example, where the reduction may seem trivial. But having millions of variables and constraints (edges) may make the reduction nontrivial.



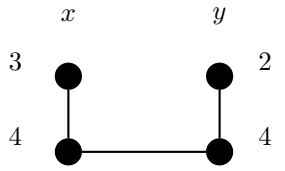
(a) A hard conflict graph. $3 \leq x \leq 5$, $1 \leq y \leq 6$, and y even. The x- and y-variables must take a value, which forms equality set constraints that translates to cliques of the conflict graph.



(b) Adding the constraint $x + y \leq 7$.



(c) Adding the constraint $|x - y| \leq 2$.



(d) Remove vertices that are neighbors of the equality-set cliques.

Figure 1 Example of clique reduction of an Integer Program with two variables and two constraints.

3. ITC 2019 conflict graphs

We introduce the conflict graphs for the ITC 2019 problem. The decision variables for the ITC 2019 are the assignment of times and rooms to classes and the enrollment of students into classes. The enrollment of students is always feasible no matter the timetable generated. Thus, we will consider the conflict graph of the class assignment variables, $x_{c,t,r}$. Also, the problem enables the use of other conflict graphs. Since an infeasible time assignment of the classes cannot become feasible by the room assignment, we also consider the conflict graph of the class-time and class-room assignments. The three graphs are denoted the class-time-room, class-time, and class-room conflict graphs. Since the complexity of graph algorithms, in general, is related to the size of the graph and the class-time-room conflict graph is the largest of the three, we examine the other two first.

The class-time conflict graph is constructed from the possible time assignments to classes. The edges are all related to the constraints considering the time assignments. The constraint types are seen in Table 1. The class-room conflict graph is constructed from the possible assignments of rooms to classes. The edges are all related to the constraints considering the room assignments. The constraint types are seen in Table 2. The class-time-room graph is generated from the two previous graphs such that edges between two class-time or class-room vertices translate into edges in the class-time-room graph. Additionally, the class-time-room graph does not include vertices that are infeasible due to the time overlapping a room's unavailability. Furthermore, it has edges for the `SameAttendees` constraints that consider both the time and room assignment. The class-time-room graph also includes edges for room double-bookings, such that two vertices with equal rooms are connected if the times overlap.

Constraint	Opposite	Time	Day	Week	Room	Pairs
SameStart		✓	-	-	-	✓
SameTime	DifferentTime	✓	-	-	-	✓
SameDays	DifferentDays	-	✓	-	-	✓
SameWeeks	DifferentWeeks	-	-	✓	-	✓
SameRoom	DifferentRoom	-	-	-	✓	✓
Overlap	NotOverlap	✓	✓	✓	-	✓
SameAttendees		✓	✓	✓	✓	✓
Precedence		✓	✓	✓	-	✓
Workday(S)		✓	✓	✓	-	✓
MinGap(G)		✓	✓	✓	-	✓
MaxDays(D)		-	✓	-	-	days over D
MaxDayLoad(S)		✓	✓	✓	-	slots over S
MaxBreaks(R,S)		✓	✓	✓	-	breaks over R
MaxBlock(M,S)		✓	✓	✓	-	blocks over M

Table 1 Table of distribution constraints of [14]. Green marks the distribution constraints providing edges to the class-time conflict graph. The `SameRoom` is not directly related to time, but classes scheduled in the same room cannot be scheduled at overlapping times. The `SameAttendees` prevents an overlap of times, which is included in the conflict graph. The four bottom constraints are only adding edges for days and weeks where only two classes are possible to violate, such that they are checked pairwise for those specific days.

Constraint	Opposite	Time	Day	Week	Room	Pairs
SameStart		✓	-	-	-	✓
SameTime	DifferentTime	✓	-	-	-	✓
SameDays	DifferentDays	-	✓	-	-	✓
SameWeeks	DifferentWeeks	-	-	✓	-	✓
SameRoom	DifferentRoom	-	-	-	✓	✓
Overlap	NotOverlap	✓	✓	✓	-	✓
SameAttendees		✓	✓	✓	✓	✓
Precedence		✓	✓	✓	-	✓
Workday(S)		✓	✓	✓	-	✓
MinGap(G)		✓	✓	✓	-	✓
MaxDays(D)		-	✓	-	-	days over D
MaxDayLoad(S)		✓	✓	✓	-	slots over S
MaxBreaks(R,S)		✓	✓	✓	-	breaks over R
MaxBlock(M,S)		✓	✓	✓	-	blocks over M

Table 2 Table of distribution constraints of [14]. Green marks the distribution constraints providing edges to the class-room conflict graph. The `Overlap` is not directly related to rooms, but classes scheduled at the overlapping times cannot be scheduled in the same room.

4. ITC 2019 clique reduction

The first reduction procedure is an improved algorithm of the clique reduction algorithm presented by Holm et al. [11]. The procedure takes advantage of the property that all classes must be scheduled. Thus, we have a clique of the vertices of the same class, where one must be chosen in a feasible solution. We denote such cliques ω_c. Any vertex adjacent to all vertices of ω_c must be of another class $c' \neq c$ and can thereby be removed. We define the set $\mathcal{V}_{\omega_c}$ as the set of vertices that are a common neighbor of all the vertices of ω_c. The clique reduction procedure is shown in Algorithm 1. Notice that since we loop all classes $c \in \mathcal{C}$, we also accept 'cliques' of size 1, i.e., a fixed vertex, which means that we include both reduction methods proposed by [11] in Algorithm 1. As reductions changes the conflict graph, they might lead to other reductions. Algorithm 2 runs Algorithm 1 until no more reductions are made.

Algorithm 1 Clique reduction algorithm

procedure ReduceClique(G)
input: Conflict graph G
1: **for all** $c \in \mathcal{C}$ **do**
2: $\mathcal{V}_{\omega_c}$ = Set of vertices adjacent to ω_c
3: Remove vertices $\mathcal{V}_{\omega_c}$
4: **return** number of removed vertices

Algorithm 2 Reduction algorithm

procedure Reduce(G)
input: Conflict graph G
1: **repeat**
2: n = ReduceClique(G)
3: **until** $n = 0$
4: **return** number of removed vertices

5. ITC 2019 SAT reduction

This section introduces a method aiming to prove a vertex v cannot be in an independent set of size $|\mathcal{C}|$. The procedure is based on the *SAT-filtering* of the branch-and-filter algorithm by San Segundo et al. [17]. [17] present an exact algorithm for binary constraint satisfaction problem and introduces filtering methods to reduce the problem size. These filtering methods are related to the reduction of our conflict graphs, and we translate the *SAT-filtering* method into a method for our conflict graphs.

Examine the graph G', which is a copy of the conflict graph G. First, color a vertex v red in G', which indicates that the variable is fixed to 1. Then mark its neighbors grey in G', which implies the variables are set to 0 since they are not feasible with v fixed. We check if this has resulted in any new classes with only one uncolored vertex. If we find a class with a single uncolored vertex v', we color v' red, and the neighbors of v' are colored grey. We continue this procedure until we cannot do any more coloring or we have colored all vertices of a class grey. If we cannot color any more vertices, we can conclude that at least one $|\mathcal{C}|$-independent set containing v exists. If we color all vertices of a class grey, fixing vertex v made the problem infeasible, and vertex v can be removed from G. We denote the method SAT reduction, and the algorithm is presented in Algorithm 3.

Algorithm 3 SAT reduction algorithm

procedure ReduceSAT(G)
input: Conflict graph G
1: Initialize set of vertices to remove $\mathcal{V}^{\text{remove}}$
2: **for all** $v \in \mathcal{V}$ **do**
3: **if** CanReduceSAT(G, v) **then**
4: add v to $\mathcal{V}^{\text{remove}}$
5: remove $\mathcal{V}^{\text{remove}}$ from G
6: **return** number of removed vertices

Algorithm 4 SAT reduction of vertex

procedure CanReduceSAT(G, v)
input: Conflict graph G, a vertex v
1: Initialize singleVertices = $\{v\}$
2: **while** singleVertices not empty **do**
3: v_i = next element of singleVertices
4: Color v_i red
5: Color all neighbors of v_i grey
6: **if** a class has one uncolored vertex v' **then**
7: add v' to singleVertices
8: **if** a class has all vertices colored grey **then**
9: **return** true
10: **return** false

The SAT reduction algorithm is expected to be slower than the clique reduction algorithm since it runs through all vertices, whereas the clique reduction runs through all class cliques. We create a reduction algorithm that first uses the clique reduction until no more vertices are removed. Then the SAT reduction is run. If the SAT reduction reduces no vertices, we stop. Else, we start the clique reductions again. The pseudo-code for our reduction algorithm with SAT reduction is shown in Algorithm 5.

Algorithm 5 Reduction algorithm with SAT

```
    procedure ReduceWithSAT(G)
    input: Conflict graph G
1:  while true do
2:      n = ReduceClique(G)
3:      if n = 0 then
4:          n = ReduceSAT(G)
5:          if n = 0 then
6:              break
7:  return number of removed vertices
```

6. ITC 2019 distribution constraints

The competition datasets also contain redundant information in the distribution constraints. Holm et al. [11] introduced various ways of finding redundant distribution constraints, including distribution constraints considering only one class or a soft constraint with a zero penalty. Other measures generate the conflict graph for the distribution constraint and check if the graph is edge-less, which means that the constraint can never be violated. We add to this procedure by combining two distribution constraints of the same type considering the same set of classes into one. If they are hard constraints, one is removed. If they are soft, the penalties are added to one and the other removed. We also include two more distribution constraint types, `MaxBlock` and `SameAttendees` in the conflict graphs. Furthermore, we check the conflict graphs for each soft distribution constraint and remove the constraint if there are no possible violations. The removal of distribution constraints changes only the size and number of entries of the instances files. Thus, removing redundant distribution constraints saves mostly computation time in reading and loading the data. Table 3 compare the number of distribution constraints removed by Holm et al. [11] and the new reduced datasets.

Instance	Total	Removed by Holm et al. [11]	Removed by proposed method
agh-fis-spr17	1,223	44	100
agh-ggis-spr17	2,690	118	179
bet-fal17	1,251	92	112
iku-fal17	2,903	245	515
mary-spr17	3,951	72	78
muni-fi-spr16	740	74	81
muni-fsps-spr17	400	79	86
muni-pdf-spr16c	2,028	231	301
pu-llr-spr17	645	118	170
tg-fal17	503	301	302
agh-ggos-spr17	1,689	75	88
agh-h-spr17	399	27	34
lums-spr18	518	28	29
muni-fi-spr17	710	96	99
muni-fsps-spr17c	709	95	108
muni-pdf-spr16	1,012	267	302
nbi-spr18	596	8	8
pu-d5-spr17	1,535	211	128
pu-proj-fal19	7,837	805	1,043
yach-fal17	645	58	67
agh-fal17	7,158	295	613
bet-spr18	1,418	110	137
iku-spr18	3,488	308	651
lums-fal17	599	51	51
mary-fal18	515	99	128
muni-fi-fal17	798	87	95
muni-fspsx-fal17	1,361	182	243
muni-pdfx-fal17	3,501	628	782
pu-d9-fal19	2,755	413	506
tg-spr18	426	236	240

Table 3 The total number of distribution constraints before reductions, the number of reductions by Holm et al. [11] and the proposed method in this paper.

7. The ITC 2019 reduction algorithm

The class-time and class-room conflict graphs are small compared to the class-time-room conflicts graphs, so we have only considered the two so far. But some of the smaller instances have class-time-room conflict graphs comparable to the class-time conflict graphs of the large instances. So we examine for which instances it is possible to create a class-time-room conflict graph and reduce it with Algorithm 2. Table 4 presents the sizes of the class-time-room conflict graphs and the memory used to create and reduce them.

Instance	Vertices	Edges	Memory (GB RAM)
agh-fis-spr17	1,274,089	-	MAX
agh-ggis-spr17	233,201	78,570,233	19
bet-fal17	506,072	960,488,910	82
iku-fal17	3,161,284	-	MAX
mary-spr17	165,428	71,375,202	12
muni-fi-spr16	36,769	6,473,241	2
muni-fsps-spr17	28,096	5,099,147	3
muni-pdf-spr16c	2,054,454	9,004,064,040	715
pu-llr-spr17	140,800	71,887,003	15
tg-fal17	51,337	7,842,469	2
agh-ggos-spr17	1,119,800	4,627,000,994	389
agh-h-spr17	1,094,730	-	MAX
lums-spr18	377,835	641,217,086	53
muni-fi-spr17	47,068	13,340,471	2
muni-fsps-spr17c	334,138	395,458,964	36
muni-pdf-spr16	848,511	1,153,342,974	97
nbi-spr18	139,006	48,376,580	11
pu-d5-spr17	72,248	10,768,023	2
pu-proj-fal19	1,640,219	3,692,303,880	310
yach-fal17	86,336	42,327,471	6
agh-fal17	3,178,415	-	MAX
bet-spr18	536,203	824,978,237	69
iku-spr18	2,824,803	-	MAX
lums-fal17	375,467	623,533,032	50
mary-fal18	179,123	85,340,122	14
muni-fi-fal17	39,255	5,902,543	1
muni-fspsx-fal17	390,476	404,496,684	38
muni-pdfx-fal17	3,631,256	-	MAX
pu-d9-fal19	585,557	403,835,840	38
tg-spr18	42,642	13,354,026	3

Table 4 Sizes of the class-time-room conflict graphs before reductions and the memory used to store and reduce them. The maximum available memory is 717 GB RAM.

Table 4 shows that with 717 GB RAM available, we are able to handle conflict graphs of 24 out of 30 instances. The six remaining instances run out of memory when generating the edges. Table 4 also shows that even a graph with 2 million vertices and 9 billion edges is reduced. This led to the investigation of reducing the graph sizes and making the graphs lighter in memory. We start by removing all edges from vertex pairs of the same class. We know these exist, but they are not used in Algorithm 2. Since we implement the edges as adjacency lists, each edge is represented twice. We introduce the concept that an edge is only found in the vertex's adjacency list with the smaller class ID. When we have to check for neighbors of a vertex, we must keep this in mind. The results show that we now can generate and reduce the class-time-conflict graphs for all the instances. We show the comparison between the implementations in Table 5. We add the class-time-room reduction to our reduction algorithm. Together with the reduction methods for the distribution constraints and the removal of unused rooms, this constitutes the proposed reduction algorithm, Algorithm 6. The class-time-room reductions can reduce a lot of vertices, but only when a time or room is completely removed from a class can it be saved in the ITC 2019 data format.

Algorithm 6 Reduction algorithm

procedure REDUCE(*inst*)
input: an instance *inst*
1: REDUCEDISTRIBUTIONCONSTRAINTS()
2: Generate $G_{c,t}$
3: REDUCEWITHSAT($G_{c,t}$)
4: Generate $G_{c,r}$
5: REDUCEWITHSAT($G_{c,r}$)
6: Generate $G_{c,t,r}$ from $G_{c,t}$ and $G_{c,r}$
7: CLIQUEREDUCE($G_{c,t,r}$)
8: **for all** $c \in \mathcal{C}$ **do**
9: c.availableTimes = distinct times of $\mathcal{V}_c(G_{c,t,r})$
10: c.availableRooms = distinct rooms of $\mathcal{V}_c(G_{c,t,r})$
11: REMOVEUNUSEDROOMS()

Instance	Vertices	Standard implementation		Memory efficient implementation	
		Edges	Memory (GB RAM)	Arcs	Memory (GB RAM)
agh-fis-spr17	1,274,089	-	MAX	786,492,529	92
agh-ggis-spr17	233,201	78,570,233	19	21,894,889	13
bet-fal17	506,072	960,488,910	82	326,415,217	33
iku-fal17	3,161,284	-	MAX	4,642,262,907	419
mary-spr17	165,428	71,375,202	12	18,504,767	2
muni-fi-spr16	36,769	6,473,241	2	1,589,565	3
muni-fsps-spr17	28,096	5,099,147	3	1,142,634	3
muni-pdf-spr16c	2,054,454	9,004,064,040	715	2,553,752,266	232
pu-llr-spr17	140,800	71,887,003	15	9,359,465	2
tg-fal17	51,337	7,842,469	2	1,398,618	2
agh-ggos-spr17	1,119,800	4,627,000,994	389	731,099,644	75
agh-h-spr17	1,094,730	-	MAX	747,000,185	117
lums-spr18	377,835	641,217,086	53	190,400,773	27
muni-fi-spr17	47,068	13,340,471	2	3,238,148	4
muni-fsps-spr17c	334,138	395,458,964	36	119,095,057	21
muni-pdf-spr16	848,511	1,153,342,974	97	295,177,971	37
nbi-spr18	139,006	48,376,580	11	11,121,004	11
pu-d5-spr17	72,236	10,768,023	2	3,103,573	3
pu-proj-fal19	1,640,219	3,692,303,880	310	411,186,017	47
yach-fal17	86,336	42,327,471	6	11,248,801	3
agh-fal17	3,178,415	-	MAX	1,824,439,021	191
bet-spr18	536,203	824,978,237	69	273,245,789	31
iku-spr18	2,824,803	-	MAX	3,934,800,679	357
lums-fal17	375,467	623,533,032	50	194,148,478	25
mary-fal18	179,123	85,340,122	14	21,029,877	10
muni-fi-fal17	39,255	5,902,543	1	1,471,143	1
muni-fspsx-fal17	390,476	404,496,684	38	127,648,642	22
muni-pdfx-fal17	3,631,256	-	MAX	4,078,112,594	362
pu-d9-fal19	585,557	403,835,840	38	62,521,341	17
tg-spr18	42,642	13,354,026	3	1,279,808	3

Table 5 The sizes of the class-time-room conflict graphs and the memory used to store and reduce them for the two different implementation forms. The maximum memory available is 717 GB RAM. Edges of the standard implementation are stored twice in the adjacency lists, as A is a neighbor of B and B is a neighbor of A. Each edge in the memory efficient implementation is only stored once; as an arc.

8. Results

This section presents the results of the reduction methods introduced. We present a comparison between the datasets produced by the algorithm of Holm et al. [11] and Algorithm 2 in Table 6. The reductions have been made on the class-time and class-room conflict graphs. The table shows that Algorithm 2 outperforms the reduction algorithm by Holm et al. [11]. The decrease in running time was expected as we no longer rely on a maximal clique cover, which is a computationally hard problem. An increase of removed vertices was also expected as we now examine the class cliques of the conflict graph systematically. Overall we see 30% more removed vertices in 43% less time. One instance *mary-spr17* showed a large percentage increase in running time, from 27 to 144 seconds, which in practice does not make much difference.

Instance	Holm et al. [11]		Algorithm 2		Difference		
	Removed vertices	Time (sec)	Removed vertices	Time (sec)	Removed vertices	Removed vertices	Time
agh-fis-spr17	11,167	13,534	11,535	651	368	3%	-95%
agh-ggis-spr17	3,656	98	4,360	53	704	19%	-45%
bet-fal17	369	32	433	18	64	17%	-44%
iku-fal17	8,616	442	10,886	120	2,270	26%	-73%
mary-spr17	117	27	147	144	30	26%	430%
muni-fi-spr16	878	7	988	4	110	13%	-41%
muni-fsps-spr17	650	15	700	5	50	8%	-66%
muni-pdf-spr16c	23,896	1,051	33,235	275	9,339	39%	-74%
pu-llr-spr17	38	19	41	15	3	8%	-20%
tg-fal17	4,997	34	5,323	16	326	7%	-52%
agh-ggos-spr17	6,458	4,449	6,869	145	411	6%	-97%
agh-h-spr17	832	42,328	3,914	1,526	3,082	370%	-96%
lums-spr18	625	134	625	9	0	0%	-93%
muni-fi-spr17	506	17	554	4	48	9%	-77%
muni-fsps-spr17c	5,131	584	7,890	86	2,759	54%	-85%
muni-pdf-spr16	1,185	80	1,430	32	245	21%	-60%
nbi-spr18	721	20	750	10	29	4%	-49%
pu-d5-spr17	746	10	912	11	166	22%	6%
pu-proj-fal19	3,539	644	3,882	642	343	10%	0%
yach-fal17	604	14	604	7	0	0%	-48%
agh-fal17	15,973	6,105	18,390	1,092	2,417	15%	-82%
bet-spr18	548	42	673	21	125	23%	-51%
iku-spr18	15,891	423	18,342	137	2,451	15%	-68%
lums-fal17	340	63	366	12	26	8%	-82%
mary-fal18	258	10	350	7	92	36%	-26%
muni-fi-fal17	297	4	328	3	31	10%	-10%
muni-fspsx-fal17	6,364	1,088	11,176	137	4,812	76%	-87%
muni-pdfx-fal17	30,719	2,148	43,318	461	12,599	41%	-79%
pu-d9-fal19	1,310	101	1,462	79	152	12%	-22%
tg-spr18	5,893	44	5,995	13	102	2%	-69%
Average						30%	-42%

Table 6 The reduction method proposed by Holm et al. [11] compared to the clique reduction algorithm, Algorithm 2. Both algorithms considers the same graph theoretic approach; to remove vertices that are adjacent to set equality cliques.

We introduced the SAT reductions by Algorithm 5 and compare with Algorithm 2. The results are presented in Table 7. Again, we reduced the class-time and class-room graphs. From Table 7 we see a further reduction of an average of 3%, but it takes on average 6.5 times longer. In a practical setting, it might not seem useful.

Instance	Algorithm 2		Algorithm 5		Difference		
	Removed vertices	Time (sec)	Removed vertices	Time (sec)	Removed vertices	Removed vertices	Time
agh-fis-spr17	11,535	638	11,608	3,627	73	1%	469%
agh-ggis-spr17	4,360	54	4,552	320	192	4%	498%
bet-fal17	433	18	566	100	133	31%	453%
iku-fal17	10,886	121	10,950	1,795	64	1%	1,383%
mary-spr17	147	142	169	166	22	15%	18%
muni-fi-spr16	988	4	988	9	0	0%	127%
muni-fsps-spr17	700	5	709	17	9	1%	220%
muni-pdf-spr16c	33,235	276	33,235	1,727	0	0%	526%
pu-llr-spr17	41	15	41	34	0	0%	122%
tg-fal17	5,323	16	5,392	55	69	1%	238%
agh-ggos-spr17	6,869	146	6,886	1,513	17	0%	933%
agh-h-spr17	3,914	1,569	3,924	3,359	10	0%	114%
lums-spr18	625	9	625	48	0	0%	406%
muni-fi-spr17	554	4	556	14	2	0%	262%
muni-fsps-spr17c	7,890	86	7,893	718	3	0%	738%
muni-pdf-spr16	1,430	31	1,446	792	16	1%	2,469%
nbi-spr18	750	10	750	66	0	0%	541%
pu-d5-spr17	913	11	937	33	24	3%	209%
pu-proj-fal19	3,882	636	4,227	3,581	345	9%	463%
yach-fal17	604	7	637	48	33	5%	548%
agh-fal17	18,390	1,099	18,847	36,848	457	2%	3,253%
bet-spr18	673	21	703	156	30	4%	653%
iku-spr18	18,342	137	18,455	1,487	113	1%	984%
lums-fal17	366	11	366	53	0	0%	361%
mary-fal18	350	7	350	25	0	0%	253%
muni-fi-fal17	328	4	329	12	1	0%	231%
muni-fspsx-fal17	11,176	137	11,186	1,265	10	0%	822%
muni-pdfx-fal17	43,318	463	43,408	7,615	90	0%	1,544%
pu-d9-fal19	1,462	75	1,510	500	48	3%	564%
tg-spr18	5,995	14	5,997	30	2	0%	123%
Average						3%	651%

Table 7 The effect of adding the SAT reduction.

We compare the reduction algorithm that introduced the SAT reduction, Algorithm 5, with our proposed reduction, Algorithm 6. Algorithm 6 considers both clique reduction and SAT reduction of the class-time and classroom conflict graphs and then uses the two reduced graphs to construct a class-time-room graph which is reduced by the clique method. The comparison is shown in Table 8. Again, the time used is around six times longer than the previous method, but this time, the number of reductions is much more than what we see from adding the SAT reductions. On average we reduce 193% more variables compared to Algorithm 5.

Instance	Algorithm 5		Algorithm 6		Difference		
	Removed vertices	Time (sec)	Removed times/rooms	Time (sec)	Removed times/rooms	Removed times/rooms	Time
agh-fis-spr17	11,608	3,627	15,314	7,796	3,706	32%	115%
agh-ggis-spr17	4,552	320	11,305	499	6,753	148%	56%
bet-fal17	566	100	3,462	2,259	2,896	512%	2,155%
iku-fal17	10,950	1,795	12,290	58,662	1,340	12%	3,168%
mary-spr17	169	166	374	242	205	121%	45%
muni-fi-spr16	988	9	1,605	22	617	62%	138%
muni-fsps-spr17	709	17	2,203	25	1,494	211%	51%
muni-pdf-spr16c	33,235	1,727	36,363	15,820	3,128	9%	816%
pu-llr-spr17	41	34	1,025	103	984	2,400%	204%
tg-fal17	5,392	55	6,316	70	924	17%	26%
agh-ggos-spr17	6,886	1,513	9,518	8,099	2,632	38%	435%
agh-h-spr17	3,924	3,359	8,259	7,447	4,335	110%	122%
lums-spr18	625	48	1,661	625	1,036	166%	1,206%
muni-fi-spr17	556	14	1,498	30	942	169%	115%
muni-fsps-spr17c	7,893	718	14,419	1,171	6,526	83%	63%
muni-pdf-spr16	1,446	792	1,780	3,300	334	23%	317%
nbi-spr18	750	66	1,389	127	639	85%	94%
pu-d5-spr17	937	33	1,556	62	619	66%	86%
pu-proj-fal19	4,227	3,581	11,951	17,866	7,724	183%	399%
yach-fal17	637	48	810	88	173	27%	82%
agh-fal17	18,847	36,848	41,500	70,762	22,653	120%	92%
bet-spr18	703	156	3,408	1,612	2,705	385%	932%
iku-spr18	18,455	1,487	21,062	58,284	2,607	14%	3,819%
lums-fal17	366	53	1,000	665	634	173%	1,161%
mary-fal18	350	25	644	143	294	84%	475%
muni-fi-fal17	329	12	1,319	23	990	301%	92%
muni-fspsx-fal17	11,186	1,265	19,869	2,124	8,683	78%	68%
muni-pdfx-fal17	43,408	7,615	57,926	66,001	14,518	33%	767%
pu-d9-fal19	1,510	500	3,126	1,433	1,616	107%	186%
tg-spr18	5,997	30	6,939	45	942	16%	50%
Average						193%	578%

Table 8 The effect of adding the class-time-room conflict graph reduction. The reduction of the class-time-room conflict graph also removes vertices that are infeasible for a single time-room pair, which does not lead to a removal of a time or a room. Thus, we compare the number of times and rooms removed from a class, i.e., vertices in the class-time and class-room conflict graph.

To see the effects of the different reduction algorithms, we construct the MIP proposed by Holm et al. [11] and compare the number of $x_{c,t,r}$ variables, i.e., the number of possible class-time-room assignments. Table 9 presents the number of variables and the percentage reductions compared to the original instances.

Instance	Original	Holm et al. [11]		Algorithm 2		Algorithm 5		Algorithm 6	
	$x_{c,t,r}$	$x_{c,t,r}$	Red. (%)	$x_{c,t,r}$	Red. (%)	$x_{c,t,r}$	Red. (%)	$x_{c,t,r}$	Red. (%)
agh-fis-spr17	1,402,094	1,275,185	9.05%	1,274,192	9.12%	1,274,089	9.13%	1,272,643	9.23%
agh-ggis-spr17	258,356	237,629	8.02%	234,813	9.11%	233,010	9.81%	232,554	9.99%
bet-fal17	519,355	509,862	1.83%	508,620	2.07%	506,072	2.56%	501,792	3.38%
iku-fal17	3,567,450	3,265,608	8.46%	3,163,376	11.33%	3,161,284	11.39%	3,159,356	11.44%
mary-spr17	166,880	165,602	0.77%	165,519	0.82%	165,428	0.87%	165,413	0.88%
muni-fi-spr16	41,355	37,150	10.17%	36,762	11.11%	36,762	11.11%	36,586	11.53%
muni-fsps-spr17	30,083	28,320	5.86%	27,895	7.27%	27,864	7.38%	27,822	7.52%
muni-pdf-spr16c	2,666,332	2,241,821	15.92%	2,054,454	22.95%	2,054,454	22.95%	2,054,431	22.95%
pu-llr-spr17	141,323	140,805	0.37%	140,800	0.37%	140,800	0.37%	140,070	0.89%
tg-fal17	67,129	53,195	20.76%	51,601	23.13%	51,337	23.52%	51,100	23.88%
agh-ggos-spr17	1,161,859	1,121,462	3.48%	1,119,979	3.60%	1,119,800	3.62%	1,119,020	3.69%
agh-h-spr17	1,145,755	1,116,988	2.51%	1,094,764	4.45%	1,094,730	4.45%	1,093,962	4.52%
lums-spr18	388,720	376,763	3.08%	376,763	3.08%	376,763	3.08%	376,751	3.08%
muni-fi-spr17	50,123	47,227	5.78%	47,078	6.08%	47,068	6.10%	47,030	6.17%
muni-fsps-spr17c	385,124	353,629	8.18%	334,156	13.23%	334,138	13.24%	333,247	13.47%
muni-pdf-spr16	877,678	853,637	2.74%	848,477	3.33%	848,461	3.33%	848,362	3.34%
nbi-spr18	141,714	139,035	1.89%	139,006	1.91%	139,006	1.91%	138,647	2.16%
pu-d5-spr17	78,589	74,201	5.58%	72,401	7.87%	72,248	8.07%	72,026	8.35%
pu-proj-fal19	1,688,728	1,644,866	2.60%	1,639,148	2.94%	1,636,305	3.10%	1,629,474	3.51%
yach-fal17	89,968	86,468	3.89%	86,468	3.89%	86,336	4.04%	86,012	4.40%
agh-fal17	3,357,610	3,206,557	4.50%	3,189,663	5.00%	3,177,108	5.38%	3,172,860	5.50%
bet-spr18	549,396	538,222	2.03%	536,336	2.38%	535,469	2.53%	535,415	2.54%
iku-spr18	3,496,473	2,913,273	16.68%	2,825,499	19.19%	2,823,669	19.24%	2,820,671	19.33%
lums-fal17	379,703	373,063	1.75%	372,619	1.87%	372,619	1.87%	372,619	1.87%
mary-fal18	183,944	179,307	2.52%	179,123	2.62%	179,123	2.62%	179,008	2.68%
muni-fi-fal17	40,649	39,403	3.07%	39,258	3.42%	39,255	3.43%	39,138	3.72%
muni-fspsx-fal17	447,914	415,672	7.20%	390,476	12.82%	390,457	12.83%	390,262	12.87%
muni-pdfx-fal17	4,500,895	3,879,342	13.81%	3,630,327	19.34%	3,627,381	19.41%	3,625,772	19.44%
pu-d9-fal19	611,688	583,836	4.55%	580,350	5.12%	579,324	5.29%	577,298	5.62%
tg-spr18	69,895	42,321	39.45%	41,815	40.17%	41,801	40.19%	41,642	40.42%
Average			7.22%		8.65%		8.76%		8.95%

Table 9 The number of $x_{c,t,r}$ variables in the MIP model when using different reduced instances. Algorithm 2 is the the improved version of the method by Holm et al. [11], Algorithm 5 uses the SAT reductions, and Algorithm 6 additionally considers the class-time-room conflict graph. All instances are compared to the reduced instances by Holm et al. [11].

9. Data availability statement

This paper shows a reduction algorithm for the ITC 2019 datasets. The original datasets can be found on the competition website `https://www.itc2019.org/instances/all` (requires log-in). The datasets produced by the reduction algorithm of this paper (Algorithm 6) is denoted the 'post-competition reduced datasets' and are available in the DTU Data repository, `http://doi.org/10.11583/DTU.20014070` ([9]). The reduced datasets used by Holm et al. [11] during the ITC 2019 are denoted 'competition reduced datasets' and are available in the DTU Data repository, `https://doi.org/10.11583/DTU.20014127` ([10]). Datasets produced by Algorithms 2 and 5 are available from the corresponding author on reasonable request.

10. Concluding Remarks

The reduction methods show to reduce the data instances of Holm et al. [11] resulting in an average decrease of $x_{c,t,r}$ decision variables of 1.73%. Holm et al. [11] already presented a 7.22% reduction of $x_{c,t,r}$ variables compared to the model using the original datasets. If we compare the original instances with the Holm [9] instances, we receive a reduction of 8.95% on average. The time used to reduce the instances shows to be huge for some of the datasets. Such reduction can only be beneficial when creating many models, working with the same data over longer periods, or the variable reduction is significant enough. The use of Algorithm 2 is always preferred over the one presented by Holm et al. [11], as Algorithm 2 removes more variables faster.

To see the effects on solving the problems, Table 10 presents the average of five runs of Gurobi 9.5 solving with the different reduced datasets. On average, we see that using the new reduced instances ([9]) results in better solutions on 9 of 17 instances with solutions. Four instances reached equal solutions; three of them are proven optimal solutions[2]. The new reduced instances also result in better lower bounds and a lower gap for more instances. The time to reach the Banch-and-Bound (BnB) tree and to find the first solution also benefits from having reduced instances.

[2]See `dsumsoftware.com/itc2019` for best-known lower bounds

Instance	Using [10]					Using [9]				
	Objective	LB	Gap (%)	Time to BnB tree	Time to first solution	Objective	LB	Gap (%)	Time to BnB tree	Time to first solution
agh-fis-spr17	**8,716.4**	1,238.6	**85.77**	**13,185.6**	**209,667.8**	12,183.0	**1,252.0**	89.72	14,334.8	281,752.0
agh-ggis-spr17	71,176.0	20,028.8	71.86	**976.9**	**52,824.5**	**60,839.0**	**20,324.2**	**66.60**	1,069.2	54,341.3
bet-fal17		59,592.0	100.00	84,352.2			**59,661.0**	100.00	**64,090.5**	
iku-fal17		**17,722.0**	100.00	8,764.7			17,684.0	100.00	**4,122.0**	
mary-spr17	**14,976.8**	**14,418.2**	**3.73**	1,233.5	13,780.6	15,025.0	14,359.0	4.44	**1,086.0**	**13,366.5**
muni-fi-spr16	4,217.6	**3,508.8**	16.81	**1,435.5**	**36,144.9**	**4,147.8**	3,458.8	**16.61**	1,706.6	48,116.2
muni-fsps-spr17	868.0	867.0	0.12	119.6	769.6	868.0	867.0	0.12	**102.3**	**582.2**
muni-pdf-spr16c		14,281.2	100.00	57,527.4			**15,399.2**	100.00	**27,522.3**	
pu-llr-spr17	10,039.6	**9,952.2**	**0.87**	1,425.2	**9,292.3**	**10,039.2**	9,949.0	0.90	**1,394.0**	12,225.0
tg-fal17	4,215.0	4,215.0	0.00	13.4	55.7	4,215.0	4,215.0	0.00	**12.8**	**35.2**
agh-ggos-spr17		**1,829.2**	100.00	19,338.8			1,816.8	100.00	**18,779.8**	
agh-h-spr17		7,875.8	100.00	38,188.5			**8,347.2**	100.00	**28,026.4**	
lums-spr18	95.0	24.0	74.74	**404.0**	1,088.2	95.0	24.0	74.74	500.4	**753.5**
muni-fi-spr17	6,852.0	2,303.0	66.39	2,194.9	133,848.3	**6,346.2**	**2,317.0**	**63.45**	**1,764.1**	**91,238.9**
muni-fsps-spr17c	**153,861.2**	1,360.0	**99.12**	25,241.7	**136,060.8**	662,742.0	1,360.0	99.79	**11,445.2**	175,308.5
muni-pdf-spr16		12,897.0	100.00	16,543.4			**12,936.0**	100.00	**13,054.1**	
nbi-spr18	18,045.6	17,807.0	1.32	172.8	1,169.8	**18,019.0**	**17,807.6**	**1.18**	**143.9**	**360.0**
pu-d5-spr17		5,057.0	100.00	39,769.5			**5,092.0**	100.00	**34,593.3**	
pu-proj-fal19										
yach-fal17	11,153.0	526.0	95.28	966.7	28,169.5	**5,201.8**	526.0	**87.61**	**648.9**	**14,947.6**
agh-fal17		6,511.8	100.00	358,719.7			**6,522.0**	100.00	**190,296.8**	
bet-spr18		61,768.0	100.00	166,946.6			**71,990.0**	100.00	**117,675.3**	
iku-spr18	26,075*	25,447.8	2.37	6,479.4	358,458.6	**25,895.2**	**25,520.0**	**1.45**	**4,802.5**	**341,164.3**
lums-fal17	353.0	252.0	28.61	849.8	**8,274.0**	**349.0**	252.0	**27.79**	**760.1**	9,720.4
mary-fal18	55,840.0	3,455.0	93.81	13,495.6	**135,659.3**	**13,477.2**	**3,459.0**	**74.30**	**12,004.4**	181,079.2
muni-fi-fal17	**14,299.6**	1,649.0	**88.47**	3,510.1	142,098.7	16,324.2	**1,654.0**	89.86	**2,722.1**	**101,995.0**
muni-fspsx-fal17		6,101.0	100.00	114,672.9			**7,855.5**	100.00	**62,889.9**	
muni-pdfx-fal17		**11,733.2**								
pu-d9-fal19		29,725.0					**29,858.0**			
tg-spr18	12,704.0	12,704.0	0.00	33.7	154.1	12,704.0	12,704.0	0.00	**28.9**	**116.5**
Total	4	6	5	4	7	**9**	**16**	**8**	**23**	**10**

Table 10 The average of five runs of solving the MIP by Holm et al. [11] with reduced datasets [10] and [9] using Gurobi 9.5. Gurobi is given a five-day time limit. The best average is shown in bold. The total row summarizes the number of best values in the respective columns. * One of the five runs did not find a solution; the average objective of the other four runs is presented.

References

[1] Achterberg, T. (2007). Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20. https://doi.org/10.1016/j.disopt.2006.10.006.

[2] Achterberg, T., Bixby, R. E., Gu, Z., Rothberg, E., and Weninger, D. (2020). Presolve Reductions in Mixed Integer Programming. *INFORMS Journal on Computing*, 32(2):473–506. https://doi.org/{10.1287/ijoc.2018.0857}.

[3] Atamtürk, A., Nemhauser, G. L., and Savelsbergh, M. W. (2000). Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40–55. https://doi.org/10.1016/S0377-2217(99)00015-6.

[4] Crowder, H. P., Johnson, E. L., and Padberg, M. W. (1983). Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, 31(5):803–834. https://doi.org/10.1287/opre.31.5.803.

[5] Gamrath, G., Koch, T., Martin, A., Miltenberger, M., and Weninger, D. (2015). Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7 (4):367–398. https://doi.org/10.1007/s12532-015-0083-5.

[6] Gramm, J., Guo, J., Hüffner, F., and Niedermeier, R. (2006). Data reduction, exact, and heuristic algorithms for clique cover. In *In Proc. 8th ALENEX*, pages 86–94. SIAM.

[7] Guignard, M. and Spielberg, K. (1981). Logical Reduction Methods in Zero-One Programming—Minimal Preferred Variables. *Operations Research*, 29(1):49–74. https://doi.org/10.1287/opre.29.1.49.

[8] Hoffman, K. L. and Padberg, M. W. (1991). Improving LP-Representations of Zero-One Linear Programs for Branch-and-Cut. *INFORMS J. Comput.*, 3:121–134.

[9] Holm, D. (2022). ITC 2019 post-competition reduced datasets. Technical University of Denmark. Dataset. https://doi.org/10.11583/DTU.20014070.

[10] Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2022). ITC 2019 competition reduced datasets. Technical University of Denmark. Dataset. https://doi.org/10.11583/DTU.20014127.

[11] Holm, D., Mikkelsen, R., Sørensen, M., and Stidsen, T. (2022). A graph-based MIP formulation of the International Timetabling Competition 2019. *Journal of Scheduling*. https://doi.org/10.1007/s10951-022-00724-y.

[12] Johnson, E. L. and Padberg, M. W. (1982). Degree-two Inequalities, Clique Facets, and Biperfect Graphs. In Bachem, A., Grötschel, M., and Korte, B., editors, *Bonn Workshop on Combinatorial Optimization*, volume 66 of *North-Holland Mathematics Studies*, pages 169–187. North-Holland. https://doi.org/10.1016/S0304-0208(08)72450-2.

[13] Land, A. H. and Doig, A. G. (2010). An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer.

[14] Müller, T., Rudová, H., and Müllerová, Z. (2018). University course timetabling and International Timetabling Competition 2019. In Burke, E. K., Di Gaspero, L., McCollum, B., Musliu, N., and Özcan, E., editors, *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2018), Vienna, Austria*, pages 5–31.

[15] Nemhauser, G. and Wolsey, L. (2014). Strong valid inequalities and facets for structured integer programs, In *Integer and Combinatorial Optimization*, chapter II.2, pages 259–295. John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118627372.ch9.

[16] Nemhauser, G. L., Savelsbergh, M. W., and Sigismondi, G. C. (1994). MINTO, a mixed INTeger optimizer. *Operations Research Letters*, 15(1):47–58. https://doi.org/10.1016/0167-6377(94)90013-2.

[17] San Segundo, P., Furini, F., and León, R. (2022). A new branch-and-filter exact algorithm for binary constraint satisfaction problems. *European Journal of Operational Research*, 299 (2):448–467. https://doi.org/10.1016/j.ejor.2021.09.014.

[18] Savelsbergh, M. W. P. (1994). Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6(4):445–454. https://doi.org/10.1287/ijoc.6.4.445.