Universiteit Leiden
Opleiding Informatica

Finding Minimal Conflicting Teacher Sets
for Dutch High School Timetabling

Name: Bob Wansink

Date: 10/04/2018

1st supervisor: Dr. W.A. Kosters
2nd supervisor: Dr. J.M. de Graaf

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Finding Minimal Conflicting Teacher Sets for Dutch High School Timetabling

Bob Wansink

April 10th, 2018

Contents

Abstract

The Dutch high school TimeTabling problem (HSTT) consists of scheduling courses in a discrete timeframe such that no conflicts exist for students, teachers, and classrooms. A Minimal Conflicting Teacher (MCT) is a set of teachers for which no conflict-free solution exists, wherein each subset allows for such a solution. Finding MCT sets is non-trivial for this problem, and is currently done by hand through experience. We translate school schedule constraints to a Boolean formula in conjunctive normal form, and check these scheduling problems for satifiability using a SAT Solver. This conversion deliberately removes schedule constraints below a certain threshold, retaining only schedule constraints that are hard demands, and leaving out all other schedule constraints. An algorithm selects sets of teachers for which satifiability must be checked, rapidly exploring the vast number of possible combinations of teachers to find Minimal Conflicting Teacher sets.

Keywords: High School Timetabling, temporal scheduling, scheduling problem, high schools, Constraint Satisfaction Problem, Boolean Satifiability, Conversion, SAT Solver, schedule constraints

Chapter 1

Introduction

Government-led reforms and a shortage of teaching staff cause a significantly increased complexity in the construction of schedules for Dutch high schools compared to schedules of other European schools. A recent push by the Dutch government for increased individualization of learning trajectories, aimed at increasing overall performance of Dutch students, has led to new challenges in the construction of schedules for high schools. One of the main competitors in the market for Dutch timetabling software currently delivers automata for constructing and optimizing these schedules.

The main subject of this thesis is an attempt to schedule a high school timetable, a problem known to be NP-complete [1, 2], by modelling the scheduling problem to DIMACS CNF format and using open-source SAT Solvers to solve the problem, and if the problem is unsatifiable, attempting to find a reduced set of actors with constraints by which the scheduling problem remains unsolvable. Finding these minimal conflicting sets of constraints is non-trivial. Once such a set is found, the scheduler can make the schedule solvable by either reassigning teachers or students to other courses, or by changing the schedule demands of teachers after approval of school management and the teacher.

In order to test the solvability of different sizes of the Dutch high school scheduling problem, a timetable generator has also been implemented that allows the user to generate timetables of a user supplied size.

In Chapters 1 to 3 of this thesis, we provide an introduction into the problem, including a historic overview. Chapter 4 provides related work that could be found on this subject. Constraints found in HSTT are defined in Chapter 5. In Chapter 6, we show how a timetable can be modelled in DIMACS CNF. We list contemporary SAT Solvers in Chapter 7. In Chapter 8, show a procedure to generate artificial timetables. We explore searching for Minimal Conflicting Teacher sets through various strategies in Chapter 9. We show in Chapter 10 the results of computational experiments that have been run. Our conclusions and further areas of research are contained in Chapter 11.

This report is written as a final product of a Master's thesis at the Leiden Institute of

Advanced Computer Sciences at Leiden University, supervised by dr. W.A. Kosters and dr. J.M. de Graaf.

Chapter 2

A brief history of the scheduling problem for Dutch high schools

The Dutch secondary school was reformed in 1968 when the *"Mammoetwet"* bill passed. Education in secondary school was reformed in order to educate all students with a broad, general education, coupled with a more specialized education tailored to that student's abilities. The education of a student on secondary school was concluded with exams in a set of subjects, some of which were mandatory (such as Dutch) while others were optional (such as Physics).

Another reform was executed in 1998, in which the havo and vwo level educations were changed significantly. The last two years of education were tailored even more closely to the abilities and interests of a student, who was given the freedom to select one of four "profiles". Each profile consisted of a set of mandatory subjects and a set of optional subjects. Some schools even allowed students to select one or more extra subjects, of which participation could not be guaranteed by the school, but in which the student is examined.

At the same time, many teachers began working part-time. Often perceived as an underpaid and undervalued profession, schools found it increasingly difficult to find teachers willing to be appointed on a full-time base, and often had to resort to hiring multiple part-time teachers instead.

These two factors combined to form a challenging scheduling problem. The largely free choice of subjects by students caused a sharply increased number of non-disjoint classes, which in turn sharply reduced the number of possible schedules. Because of the large amount of quality demands placed on the schedule, this led to difficulties in finding a schedule which was acceptable for all participants. The reduced availability of teachers further limited the number of possible solutions, and due to sharp scheduling restraints imposed from collective bargaining, further decreased the number of acceptable schedules for all participants.

After several evaluations, the reform of 1998 was revised in 2007. The number of courses

that were offered by schools was reduced, and a number of courses (colloquially known as "stapelvakken') were simplified in structure. This led to slight decrease in scheduling complexity.

Schools are encouraged by the government to excel. Many of the larger schools offer special tracks for gifted students, focussing on specific areas such as technology, sport, or healthcare. Other schools offer a bilingual education (TTO), to prepare students for the more and more international society. Unfortunately, this does have significant repercussions for the complexity of the schedule, often leading to situations that seem like a "school-within-a-school".

In recent years, the VO-raad has pushed the "Leerling 2020" plan, which aims at further individualizing the student learning trajectory. It proposes making participation in even more subjects optional, and giving students more freedom in choosing what courses they wish to attend at any given moment. To accomodate for this increase in freedom of choice, schedules will need to accomodate optional attendance of courses by students. It is expected that many schools will have difficulties in realizing a schedule with contemporary school scheduling software.

Chapter 3

Problem description

We start with an input S. S is a set of courses, each of which must be scheduled on a timeslot t, consisting of a segment m, a workday d and a period t. Each course can have a number of teachers and a number of students assigned to it. The main problem in high school timetabling (HSTT) is assigning each course a timeslot in such a way that the resulting schedule of each teacher and student is collision free. Input S can have a set of constraints C imposed on it. These schedule constraints further limit the number of possible solutions.

We can convert the constraints to a CSP description in a file encoded in the DIMACS CNF format and feed it to an open source SAT Solver. The SAT Solver will attempt to determine if there is a solution possible, and will output the first found solution, if any is found. If a solution is found, this solution is then read back in, and the fitness of the schedule is measured. We use this methodology to check for satifiability of a schedule within a set of fixed constraints.

If S has been unsuccessfully scheduled with all constraints, we can remove a number of constraints and export S again. Can we repeat this process in the hopes of finding a minimal subset of constraints that cannot be satisfied as a whole in polynomial time?

3.1 Example problem

Problem S_e is a set with twelve courses that must be scheduled within three days (d_1, d_2, d_3), each containing four periods (h_1, h_2, h_3, h_4). Each course is attended by the same student (there is only one), and each subject has a specific teacher. Some subjects have constraints that require the courses to be scheduled in different days, see Table 3.1 for the requirements.

There are five additional constraints imposed on S_e.

- Subject ne must be scheduled in h_1
- Subject fa must not be scheduled in h_4

Subject	Teacher	nr. courses	nr. days
ne	A	3	3
fa	B	2	1
en	C	2	2
du	D	2	1
te	E	1	1
ak	F	2	2

Figure 3.1: Example problem

- Teacher C must not be scheduled in d_1
- Subject te must not be scheduled in d_2
- Subject ak must not be scheduled in h_3, h_4

Chapter 4

Related work

The school timetabling problem is approached from a number of different directions in recent years. The timetabling problem is a well researched problem, with many known solution approaches. One such approach used on South African school timetables in [3], uses a two-phased Genetic Algorithm approach.

Another approach is using a Late Acceptance Hill Climber, such as in [4]. A Late Acceptance Hill Climbers accepts a candidate solution if it is not worse than a solution that was "current' several steps before, instead of the last found "current' solution, allowing it to escape local optima. The distance of the "current" solution is flexible, and can be lengthened or shortened at runtime. This allows the algorithm to be more flexible at the start of its process and more robust at the end.

Demirović and Musliu have used a combination of MaxSAT and local search to solve instances of the high school time tabling problem in [5]. Constraints are partitioned into two types, hard and soft, where soft constraints are weighted. The goal is to find an assignment which satisfies all hard constraints and minimizes the sum of the weights of the unsatisfied soft constraints.

Simulated Annealing is also used as an algorithm to solve High School Timetabling (HSTT) problems in [6]. The approach here is to assign neighborhoods to every type of movement in the search space, and to measure the fitness of each solution by multiplying the number of movements in each of the neighborhoods while imposing a maximum fitness. The algorithm allows for changes to the neighborhood structures within a search. Using this approach allows an algorithm to search for solutions of only certain differences, without losing the ability to leave locally optimum solutions.

Tabu Search [7], a Global Optimization algorithm, seeks to prevent a heuristic from finding solutions that have few differences with the current optimum. It is often used with hill climbing heuristics, and compares features of the current locally optimal solution with new-found solutions. Only if the two solutions have few or no corresponding features is the solution accepted as a new optimum. This allows an automaton to step over trivial

solutions that might keep it locked on a local optimum.

No known works could be found by the author regarding the specific problem of finding minimal conflicting sets of constraints in HSTT.

Chapter 5

Constraints found in Dutch HSTT

A schedule for Dutch high schools typically contains many constraints. These constraints are imposed on the schedule by teachers, students, the board of directors, deans, and the limited availability of resources such as classrooms, audio-visual equipment, or computers. Constraints can be *hard constraints* or *soft constraints*. Hard constraints are constraints that must be met. An example of this could be the availability of a teacher, or the constraint that no first-year student is allowed to have empty timeslots between his or her first and last courses of a day (called gap hours). Soft constraints are not mandatory, but are highly desirable. An example could be the reduction of the number of gap hours in the schedule of a teacher. A field of conflict in most schools is the degree of separation between hard and soft constraints. Some hard constraints are not perceived as hard constraints by all parties, and some theoretically soft constraints are actually hard constraints in practice. The goal is finding a schedule that satisfies all hard constraints, and most soft constraints. This is the High School Timetabling Problem (HSTT).

The inability to create schedules for Dutch High Schools by hand that satisfy all constraints has led one of the main competitors on the market for Dutch timetabling software to model constraints in terms of penalty points. If the criteria of a constraint are not met, penalty points are awarded to the schedule. The total number of penalty points of a schedule, its penalty value, is its "fitness". If the penalty value is decreased, the solution is more fit. The automata try to minimize the penalty value for the schedule, thus maximizing fitness. Hard constraints are typically valued at one million penalty points. Soft constraints are valued at arbitrary values, but it is generally advised to keep the value between one hundred and ten thousand penalty points.

Even though hard constraints are imposed on a schedule, this does not mean the automata are forced to only accept solutions in which all hard constraints are met. A constructing automaton will accept any solution which does not contain conflicts. Once a solution is found containing all courses, its fitness is stored. If the automaton then finds another solution with higher fitness, it is considered to be an improvement. A solution that

does not satisfy all hard constraints might be optimized to a solution that does satisfy all hard constraints, either through a different assignment of timeslots to courses, or by changing student-to-course or teacher-to-course assignments, if such an assignment exists.

It is not always possible to satisfy all hard constraints in a schedule, and it is generally not possible to satisfy all constraints. Some constraints can be contradictory. Consider the following constraints:

1. Teacher T wants to be scheduled exclusively on Tuesdays.
2. The subject Mathematics is to be scheduled on any timeslot except those of Tuesday afternoon.
3. Classroom R is exclusively available on mornings.

If teacher T is assigned a course in Mathematics in classroom R, it will not be possible to schedule this course without breaking at least one of these constraints. Soft constraints are even less likely to be completely satisfied, because these constraints are often used to spread out courses of a specific subject-group combination over the available timeslots in the week. The mis-placement of a single course could lead to a slight decrease of fitness.

5.1 Schedule constraints

Schedule constraints are imposed upon the set of all possible timeslots in the schedule of a student or teacher. Each schedule constraint contains a tuple (R, T), where R is set of courses, and T is a set of timeslots. Schedule constraints penalize the placement of courses in R on timeslots in T. An example of this could be all courses of student s_1, and all timeslots of Monday.

Schedule constraints use predicate functions that return a value which can be used in a fitness function. An example of such a predicate function is counting the total number of available timeslots between the first and the last unavailable timeslot in T. The fitness function can then award penalty points to the schedule if it either exceeds a user-supplied value, or does not exceed it.

An example would be a constraint imposed on a teacher's schedule, specifying that no more than three available timeslots are acceptable on Fridays, with a penalty value of one thousand points for each available period after the third. If the schedule contains seven available timeslots on Friday, the constraint would add four thousand penalty points to the complete schedule.

5.2 Course constraints

There are many constraints which can be imposed on sets of courses. The model used by Dutch timetabling software currently imposes the following constraints on sets of courses:

- Educational constraints
- Concurrent course constraints
- Block constraints
- Student-to-class assignment constraints
- Timeslot-availability

Educational constraints

Educational constraints are imposed on a schedule to improve the rate in which students pick up on the subject. These constraints can be used to penalize the placement of specific courses in a schedule. The user can specify the penalty points awarded for:

- not scheduling the set of courses on the desired number of days,
- each number of free periods between two courses scheduled in the same day (gap hours),
- each number of breaks between two courses scheduled in the same day,
- an asymmetrical scheduling of the courses over all time slots,
- an asymmetrical scheduling of the courses over all days,
- scheduling a course on a specific timeslot,
- not scheduling a course on a specific timeslot,
- scheduling the set of courses in a specific block,

Each of these constraints is imposed on sets of courses with varying levels of commonness. The most global level is the total set of courses in the schedule. If a scheduler specifies a constraint on the desired number of days at this level, all courses will be scheduled in that number of days. This level is often used to supply default educational constraints. It is however possible to specify constraints for smaller sets of courses, such as all courses of a specific subject, or all courses for a specific department. If a constraint is imposed on a smaller, more specific set of courses, it overrides the constraints at any more global level for those courses.

A gap hour is a timeslot in the schedule of a student or teacher that is assigned a course while a preceeding and a succeeding timeslot are not assigned a course. If there are multiple adjacent gap hours, these are collectively referred to as a gap in that schedule. Gaps are generally unwanted, and most schools seek to minimize the number of gap hours in their schedules.

Block constraints

Block constraints are imposed on sets of courses that must be scheduled on the same day, on adjacent timeslots. This type of constraint is often applied to courses such as Physical Exercise, for which it is impractical to schedule two or more separate courses, or courses for practical, technical courses, such as Metalworking or Farming. Block constraints can be imposed on a set of any size, though smaller sets of two or three courses are the most common.

Student-to-class assignment constraints

Student-to-class assignment constraints are used when allowing changes to classes while scheduling. Sometimes, an improvement to a schedule is blocked because there is a small subset of students that cannot be scheduled at a course's preferred timeslot. If the class assignment of students can be changed at runtime, the algorithm might be able to swap one student-to-class assignment for another, allowing the course to be scheduled at its preferred timeslot. Changing student-to-class assignments is mostly allowed before the schedule is published to everyone involved. Changing student-to-class assignments after publication is often not allowed, or with a minimal number of changes. These changes can optionally be penalized with constraints.

Timeslot-overflow

Timeslot-overflow constraints impose a maximum and minimum number of courses that are allowed to be scheduled in a specific timeslot. If a given school building only contains seventeen classrooms, then scheduling eighteen courses on any single timeslot will make it impossible for one of the courses to be scheduled into a classroom. It is also possible to count the number of requested classrooms for a course, or even a user-supplied value that is dictated per course. As such, it is possible to create a schedule that makes optimal use of classrooms containing smart boards, or one that minimizes the peak demand of courses requiring the use of a computers.

It is also possible to dictate that there has to be at least one timeslot that contains a certain number of courses. This can be very useful if all courses in a specific subject and for a specific set of students are to be given at the same time. This is often used to force all courses for the subject Mentoring onto the same timeslot for students in a specific department.

5.3 Change constraints

When a schedule has been created and is running, it is possible that changes need to be made to accomodate for errors or unexpected long unavailability of teachers (e.g., disease or contract termination). Because some changes to the schedule can be quite difficult, and

the use of contemporary automatic scheduling software is preferred, it might be preferable to schedule the changes automatically.

Unfortunately, changes to a running schedule are regarded as highy unfavorable, and large unplanned changes to a schedule are often met with unrest and dislike. Changing the schedule with an automaton must therefore also be constrained by the number of changes required to find a better solution. This can be accomplished by noting down the original timeslots of each course, and counting the number of courses that have been moved. Thresholds and penalty points can be applied, forcing an automaton to keep the number of changes in check.

These constraints are solely used in schedules that are already running. As such, they will not be further treated here.

Chapter 6

Modelling a timetable in DIMACS CNF

Open source SAT Solvers use the DIMACS CNF file format[1] as input. DIMACS CNF files contain Boolean functions in Conjunctive Normal Form denoted in a standardized way. An example of a Boolean formula in conjunctive normal form is:

$$(x_1 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee x_4 \vee x_5) \wedge (\neg x_3 \vee \neg x_4)$$

Figure 6.1 shows how this formula can be denoted in simplified DIMACS CNF format.

```
1 c Sample Boolean formula.
2 1 4 -5 0
3 -1 3 4 5 0
4 -3 -4 0
```

Figure 6.1: Example DIMACS CNF file.

The first line in the file always denotes the problem as a Boolean formula in Conjunctive Normal Form. This file lists it as consisting of five literals and three clauses. Every subsequent line not starting with the letter c (which denotes comments and is ignored) then lists a new clause. Every clause is made up of a row of either positive or negative numbers. A number represents a literal in the Boolean formula. If a literal is negated in the Boolean formula, then the number representing it in the file is also negated. The literal x_3 in the Boolean formula above is for instance represented by the number 3 in the second clause, and the number -3 in the third clause of the example DIMACS CNF file. The number 0 denotes the end of each clause.

[1] see: http://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.tex

6.1 Modelling a course in DIMACS CNF

To model a timetable to a simplified DIMACS CNF file, we must convert a set of courses (each a tuple containing sets of students and teachers) and constraints to a set of literals and clauses. The literals and clauses added to the model for each course in order to assign those courses exactly one timeslot and preventing conflicting courses from being scheduled at the same timeslot are all mandatory. No solution can be considered a valiable schedule without satisfying at least these constraints. Additional constraints must be added to further limit the number of possible solutions such that the hard demands that a scheduler or the school poses to the schedule are also met. A school can therefore be modelled as a set of literals L_m, a set of constraints C_m, and a second set of literals L_a and constraints C_a, where L_m is the set of all literals representing the scheduling of a course on a timeslot, C_m is the minimal set of clauses required to schedule each course on exactly one timeslot without conflicts, and L_a and C_a are the sets of literals and clauses used to model other types of constraints, such as absence on certain dates. The schedule cannot be modelled without each of the literals and clauses in L_m and C_m, but can be scheduled with a subset of the literals and clauses in L_a and C_a (though it might then not satisfy all constraints).

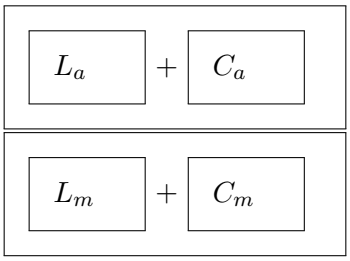

Figure 6.2: L_m and C_m are the minimal set of clauses and literals required for modelling a conflict-free schedule. L_a and C_a are additional literals and constraints on top of that minimum.

When modelling a timetable to a simplified DIMACS CNF format, each course can be represented by a set of literals L for each timeslot that course can be scheduled on. If a timetable has eight periods, five days, and one segment, then the scheduling of a course on a timeslot is represented by 40 consecutive literals $L = \{x_1, x_2, x_3, \ldots, x_{39}, x_{40}\}$ for each course. To schedule a course into a timeslot, one of the literals in this set can be made true, while all the other literals for this course are made false. This can be achieved by adding a number of clauses to the file as shown in Figure 6.3.

The clause in Figure 6.3 can only be satisfied when at least one of all the literals representing the scheduling of course c_1 on a timeslot is made true. A course must however be scheduled in exactly one timeslot. The clause above is satisfied when it contains at least one positive literal among all the literals, but does not constrain solutions from contain-

```
c Course c1 must be scheduled in at least one of these timeslots
c 1 represents the first timeslot, 2 the second, etc.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
    26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 0
```

Figure 6.3: Course c_1 is assigned a number of literals.

ing multiple true literals. In order to prevent course c_1 from being scheduled in multiple timeslots, we must add additional clauses that can only be satisfied if exactly one of the literals representing the possible timeslots for course c_1 is true. These clauses can be seen in Figure 6.4.

```
c Each combination of two literals representing possible
   timeslots for course c1 must at most contain one positive
   literal.
-1 -2 0
-1 -3 0
-1 -4 0
...
-1 -40 0
-2 -3 0
-2 -4 0
...
-38 -39 0
-38 -40 0
-39 -40 0
```

Figure 6.4: These clauses cannot be satisfied if multiple literals representing the scheduling of course c_1 on a timeslot are true.

Each unique pair of literals representing possible timeslots for course c_1 is assigned a clause that prevents a solution if both literals are positive. Combined with the first clause in Figure 6.3, this forces c_1 to be scheduled at exacly one timeslot. Thus, the number of clauses required per course to only allow solutions where that course is scheduled in exactly one timeslot is $n*(n-1)/2+1$, where n is the number of unique timeslots that course can be scheduled in.

6.2 Preventing conflicting courses from being scheduled at the same timeslot

Courses conflict if a student or teacher attends both courses at the same timeslot. To prevent two conflicting courses from being scheduled at the same timeslot, we must disallow solutions containing positive values for the corresponding literals for both courses. Figure 6.5 shows how scheduling course c_1 on any of the possible timeslots can be represented by literals 1 to 40. Scheduling course c_2 is represented by literals 41 to 80. The previous clauses disallow solutions where two literals representing the same timeslot for both courses c_1 and c_2 are both true. The number of clauses required per pair of overlapping courses is equal to the total number of timeslots.

```
1  c Because courses c1 and c2 conflict, we wish to add clauses that
       can not be satisfied if both courses are scheduled at the
     same schedule timeslot.
2  -1 -41 0
3  -2 -42 0
4  -3 -43 0
   ...
39 -38 -78 0
40 -39 -79 0
41 -40 -80 0
```

Figure 6.5: Courses c_1 and c_2 must not be scheduled at the same timeslot, label=Clauses preventing two courses from having corresponding positive literals.

6.3 Preventing gaps

Teachers and students also impose a number of additional constraints on the assignment of timeslots of their courses. Gaps for instance are disallowed in certain departments, and certain timeslots are disallowed because either the teacher or student is unavailable, or because the school does not want courses to be scheduled at that time (e.g., meetings). Because these are constraints imposed upon a combination of courses, we cannot use the literals representing the possible schedule timeslots of individual courses to impose these constraints. We must therefore assign each teacher and each student with a representation of consecutive literals much like the courses. To disallow gaps, we must add clauses for each individual gap configuration. The number of clauses added is equal to $\sum_{i=1}^{i<h-2} h - 1 - i \times d$ where h is the number of periods and d is the number of days.

In Figure 6.6, we represent the schedule of teacher T with a set of consecutive literals

valued 161 to 200. The schedule of teacher T is represented by 40 literals, each representing a unique timeslot. Because gaps only exist on a single day, and are never counted over multiple days, we construct clauses that represent timeslots for each possible gap configuration for each day, and which cannot be satified if a gap exists conforming to that clause's configuration.

```
1   c Disallow gaps in the schedule of teacher T.
2   -161 162 -163 0
3   -161 162 163 -164 0
4   -161 162 163 164 -165 0
5   -161 162 163 164 165 -166 0
6   -161 162 163 164 165 166 -167 0
7   -161 162 163 164 165 166 167 -168 0
8   -162 163 -164 0
9   -162 163 164 -165 0
10  -162 163 164 165 -166 0
    ...
178 -197 198 -199 0
179 -197 198 199 -200 0
180 -198 199 -200 0
```

Figure 6.6: Prevent gaps in schedule of teacher T.

We must however also link all the literals corresponding to the timeslots of all courses teacher T attends to the literals that correspond to the timeslots of T's schedule. We do this by appending further constraints. If teacher T attends courses c_1, c_2, c_3 and c_4, and those courses are represented by literals 1 to 40, 41 to 80, 81 to 120, and 121 to 160 respectively, then we can append the clauses shown in Figure 6.7.

Each clause pairs one of the literals corresponding to a possible timeslot assignment of a course teacher T attends to a literal corresponding to the same timeslot in teacher T's schedule. If the course's literal is true, then the corresponding literal in the schedule of teacher T must also be true.

This does however allow the solution to permit false positives, as a false value for the literal of the course is enough to satisfy the clause, and the value for the literal of the corresponding timeslot of teacher T can then be any value. Because these false positives might allow solutions with gaps to be accepted, we must append further clauses to prevent this from happening, such as in Figure 6.8

Each clause lists the literals corresponding to identical timeslots assignments of all the courses teacher T attends, plus the literal corresponding to teacher T's schedule. If none of the literals of the courses are true for that schedule timeslot, then the literal corresponding to the schedule timeslot of teacher T must be made false, or the clause would not be

```
c Link the literals of all the courses of teacher T to his or her
    schedule postions.
−1 161 0
−2 162 0
−3 163 0
...
−39 199 0
−40 200 0
−41 161 0
...
−158 198 0
−159 199 0
−160 200 0
```

Figure 6.7: Linking the timeslots of courses to the schedule of T.

```
1 41 81 121 −161 0
2 42 82 122 −162 0
3 43 83 123 −163 0
38 78 118 158 −198 0
39 79 119 159 −199 0
40 80 120 160 −200 0
```

Figure 6.8: Preventing false positives in the schedule of T.

satisfied.

6.4 Preventing up to n gaps in a schedule

Schools might not disallow gaps completely. Due to the high amount of optional subjects and the resulting number of (non-disjoint) cluster groups, a schedule completely free of gaps is most likely impossible. Schools therefore only disallow gaps for certain departments, while allowing gaps in other departments *up to a certain limit*.

If gaps are completely disallowed,then clauses are used for each possible gap configuration. To allow gaps up to a certain limit, we add a literal for each allowed gap configuration, and use those literals to disallow certain combinations. Figure 6.9 demonstrates this for up to one gap consisting of one gap hour.

If T has a schedule where three of the seven timeslots contain courses scheduled such as in Figure 6.10 and we were to allow up to three gap hours in the schedule of T, we would

```
c force additional literals to true for certain gap
   configurations
−161 162 −163 201 0
−161 162 163 −164 202 0
−161 162 163 164 −165 203 0
−161 162 163 164 165 −166 204 0
−161 162 163 164 165 166 −167 205 0
−161 162 163 164 165 166 167 −168 206 0
−162 163 −164 207 0
...
−197 198 −199 378 0
−197 198 199 −200 379 0
−198 199 −200 380 0
c disallow solutions containing more than one gap.
−202 0
−203 0
−204 0
−205 0
−206 0
−208 0
...
−379 0
```

Figure 6.9: Preventing more than two gaps in the schedule of T.

need a number of clauses such as those in Figure 6.11. Obviously, we can dispense with disallowing combinations that are impossible, such as 11 and 20.

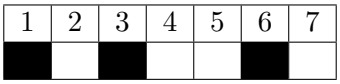

Figure 6.10: Example schedule for T. Black cells are timeslots that contain courses.

6.5 Modelling timeslot-overflow constraints

To prevent a set of courses from being scheduled in such a way that the total number of courses on a certain timeslot never exceeds a fixed amount, we will need to impose further

```
c force additional literals to true if a certain gap
    configuration exists
−1 2 −3 11 0
−2 3 −4 12 0
−3 4 −5 13 0
−4 5 −6 14 0
−5 6 −7 15 0
−1 2 3 −4 16 0
−2 3 4 −5 17 0
−3 4 5 −6 18 0
−4 5 6 −7 19 0
−1 2 3 4 −5 20 0
−2 3 4 5 −6 21 0
−3 4 5 6 −7 22 0
−1 2 3 4 5 −6 23 0
−2 3 4 5 6 −7 24 0
−1 2 3 4 5 6 −7 25 0
c disallow solutions containing more than three gap hours.
−11 −22 0
−15 −20 0
−25 0
−24 0
−23 0
```

Figure 6.11: Preventing more than four gap hours in the schedule of T.

clauses upon the solution. If we were, for example, to prevent three or more courses on period 1 of Monday from a set of five courses, we would need clauses such as in Figure 6.12. Unfortunately, these lists of clauses exponentially grow with the size of the subset of courses and the number of allowed courses. Modelling overflow constraints is therefore currently disabled.

6.6 Modelling specific schedule constraints

A schedule of a teacher, student, or even classroom can be constrained at specific timeslots. A teacher that cannot easily get out of bed might be constrained on timeslots of each first period of each day, allowing the teacher to start a bit later. To prevent a course from being scheduled on a specific timeslot, we can easily add a short clause for each disallowed timeslot by explicitly forcing the literal corresponding to the scheduling of a specific course

```
c disallow all combinations of three literals representing monday
    's first timeslot for five courses.
-1 -41 -81 0
-1 -41 -121 0
-1 -41 -161 0
-1 -81 -121 0
-1 -81 -161 0
-41 -81 -121 0
-41 -81 -161 0
-41 -121 -161 0
-81 -121 -161 0
```

Figure 6.12: Preventing timeslot-overflow in a subset of courses, label=Disallowing timeslot overflow.

on a specific timeslot in the schedule of the teacher to false. An example of disallowing a timeslot for a student or teacher is shown in Figure 6.13.

```
c disallow all combinations of the literals representing monday's
    first timeslot for each course in subset $S$ of $T$.
-81 0
```

Figure 6.13: Preventing overflow in a subset of courses in the schedule of teacher T.

Blocking off a timeslot for a teacher, student, or classroom results in n clauses, where n is the number of courses for that teacher, student, or classroom. As such, these constraints add $n \times m$ clauses, where m is the number of constrained timeslots.

6.7 Modelling aspecific schedule constraints

Most of the teachers's schedules are imposed with aspecific schedule constraints. These are constraints that disallow courses from being scheduled in a part of the schedule, without specifying which part. A teacher that has a Full-Time equivalent (FTE) of 0.7 must for instance be scheduled in only four days, leaving the fifth day available. However, which day remains unscheduled is up to the school, and the scheduler, to decide.

To prevent the courses of a teacher from being scheduled in more than four days, we need to add an extra literal to the Boolean formula for each day in the schedule of each teacher. If none of the timeslots for the corresponding day are unavailable, then that literal is forced to true with a clause such as in Figure 6.14. Further clauses must then be added, disallowing solutions where all days contain unavailable timeslots, such as in Figure 6.15

```
c if all timeslots on day x are available (false), then extra
    literal n is true.
1 2 3 4 5 6 7 8 41 0
9 10 11 12 13 14 15 16 42 0
17 18 19 20 21 22 23 24 43 0
25 26 27 28 29 30 31 32 44 0
33 34 35 36 37 38 39 40 45 0
```

Figure 6.14: Imposing aspecific schedule constraints on the schedule of teacher t.

```
c disallow a solution where all days have scheduled courses.
-41 -42 -43 -44 -45 0
```

Figure 6.15: Disallow solutions with all days scheduled.

If less than four days are to be scheduled on, we need to impose further clauses, such as in Figure 6.16.

```
c disallow a solution where more than two days contain scheduled
    courses.
-41 -42 -43 0
-41 -42 -44 0
-41 -42 -45 0
-41 -43 -44 0
-41 -43 -45 0
-41 -44 -45 0
-42 -43 -44 0
-42 -43 -45 0
-42 -44 -45 0
-43 -44 -45 0
```

Figure 6.16: Imposing more aspecific schedule constraints on the schedule of teacher t.

6.8 Disallowing previous or specific solutions

If a SAT Solver finds a solution to the problem, it can list that solution. That solution can be translated to a single clause, and can be appended to the schedule to prevent the SAT Solver from listing it as a solution. If, for example, a SAT Solver was used to solve the

sample Boolean formula at the top of this chapter, then it might have generated `SAT -1 -2 -3 4 -5` as output. We can then append an additional clause to the formula to disallow that solution by adding a clause that negates the entire solution as shown in Figure 6.17.

```
p cnf 5 3
c Sample Boolean formula.
1 4 -5 0
-1 3 4 5 0
-3 -4 0
c Disallow a specific solution:
1 2 3 -4 5 0
```

Figure 6.17: Disallowing a specific solution.

6.9 Generating CNF through a broker

Another tool that can generate DIMACS CNF files is Sugar [14]. Sugar uses a self-made intermediate language to generate DIMACS CNF files (Sugar CSP files) that can be solved using a SAT Solver such as MiniSat. An example of such a file is:

```
; domain definitions
(domain all 1 3); all timeslots
; course definitions
(int c1 all)
(int c2 all)
(int c3 all)
(int c4 all)
(int c5 2 2); fixed course scheduled in timeslot 2
(int c6 all)
(int c7 all)
(int c8 all)
(int c9 all)
; definitions of non-disjoint courses
(alldifferent c1 c2 c5)
(alldifferent c3 c6)
(alldifferent c9 c8 c7)
(alldifferent c4 c7)
; hard constraint teacher unavailable last timeslot
(and (!= c1 3) (!= c2 3))
; hard constraint timeslot availability.
```

```
21 (count 3 (c1 c2 c3 c4 c5) ge 1); timeslot 1 at least 3 courses
22 (count 2 (c6 c7 c8 c9) lt 3); timeslot 3 at most 1 courses.
```

Example Sugar CSP file

Every Sugar CSP file contains a set of DOMAIN definitions, INTEGER definitions, and CONSTRAINT definitions, which must adhere to a syntax specification created by the author of Sugar [15]. Once the file is complete, Sugar will generate a DIMACS CNF file that describes the same problem in conjunctive normal form. Sugar will automatically start a SAT Solver to try and solve the problem, and will output a solution to the problem if one is found.

A Sugar CSP file first lists a set of DOMAINS. These domains represent the available ranges of positive natural numbers that are considered valid solutions for each of the integers assigned to that domain. These domains are followed by INTEGER definitions. Each integer must be assigned a value within the domain specified. This is the solution the SAT Solver attempts to find. Integers may also be assigned a fixed value, or a self-defined range.

Below the integer definitions the CSP file defines CONSTRAINTS. There are several KEYWORDS available in the CNF syntax to facilitate easier definition of constraints. The ALLDIFFERENT keyword, for example, imposes a constraint on a set of integers that no two integers in the constraint can have identical values.

There are several different keywords available that allow for further limiting of the allowed assignments of values for each course. Combining these keywords is also possible, which allows for further limiting of the assignments.

To generate a Sugar CSP file that describes a scheduling problem, we define a domain D with a range equal to the total number of available timeslots. In a school schedule with one segment, five days and eight hours, this domain would be $(1 \ldots 40)$. Assigning an integer c_n to each of the courses, and supplying them with domain D, specifies that each course must be scheduled in a timeslot within the available timeslots. We must however also supply a large amount of constraints to prevent non-disjoint courses from being scheduled at the same time, otherwise a solution where each course is scheduled in timeslot 1 would be acceptable. To prevent overlap, the ALLDIFFERENT keyword allows us to supply a set of non-disjoint courses for each teacher and each student which must be scheduled at different timeslots. We can also combine a number of keywords to prevent assignments on specific days or periods. An example would be (AND (!= c_1 3) (!= c_2 3)) which constrains courses c_1 and c_2 from being assigned to periods 3.

Using an intermediary language allows for small optimizations by the broker. It might add additional and notably redundant clauses to the DIMACS CNF file based on internal evaluations. These additional clauses might optimize the algorithm of the SAT Solver and lead to faster solution-checking.

6.10 CNF File Statistics

CNF files generated by modelling a scheduling problem into a DIMACS CNF file are generally quite large. The number of literals can quickly reach several hundreds of thousands, and the number of clauses may reach into the millions for typical Dutch high schools, which generally contain at least eight periods, two segments, over fifty teachers with varying Full-Time Equivalents, and around 1.000 students.

Literals

Defining the timeslot of a single course within the schedule of a single actor such as a teacher or student requires pds literals, the number of all timeslots. This number is equal to the number of periods p multiplied by the number of all days d and every school segment s. The same number of literals is used to keep track of the available and unavailable timeslots of each actor. Assuming a schedule of $2,000$ courses for 100 teachers and $1,200$ students would yield $1,300 \times pds + 2,000 \times pds$ literals. If a school has 2 school segments and 40 timeslots, this would lead to total of $104,000 + 160,000 = 164,000$ literals.

Certain constraints require the use of additional literals. Each allowed gap in a schedule of a single actor for instance increases the number of literals by one. Aspecific schedule constraints also add literals.

The total number of literals can be calculated with

$$L = (T + S) \times (pds + g) + C \times pds + a \times d$$

where T is the total number of teachers, S is the total number of students, C is the total number of courses, g is the total number of literals added to prevent gaps, a is the total number of aspecific schedule constraints, and L is the total number of literals.

Clauses

Scheduling each course adds a large amount of clauses to the CNF file. One clause is added to ensure that the course is scheduled on at least one timeslot, then $n \times (n-1)/2$ further clauses are added to ensure that the course is scheduled on no more than 1 timeslot. In the example above, this would add a total of 1,999,000 clauses.

Each course must also be scheduled at the same timeslot on the same day for each actor attending that course. Assuming that about half the courses within one department of a schedule are conflicting due to students and teachers attending both courses, and each department containing about 100 courses, this would add a futher $D \times (C_d/2) \times pds$ clauses, where D is the number of departments, and C_d is the number of courses for each department. Specific schedule constraints add $n \times m$ clauses, where n is the number of courses for the teacher, student, or classroom, and m is the number of timeslots blocked.

The number of clauses required to prevent gaps is equal to $\sum_{i=1}^{i<h-2} h - 1 - i \times d$ where h is the number of periods and d is the number of days.

Chapter 7

SAT Solvers

SAT Solvers are programs used to search for solutions (=satisfying assignments) to Boolean formulas in conjunctive normal form. When run, a SAT Solver uses an algorithm to assign *true* and *false* values to literals in the fomula in order to find an assignment such that all clauses in the Boolean formula result to *true*. If such a solution is found, the SAT Solver labels the formula as *satisfiable*. If all possible combinations have been exhausted and none result in a *true* result for all the clauses in the Boolean formula, then the formula is labeled as *unsatisfiable*.

SAT Solvers are not magic. They are as constrained by NP-complete problems as all other algorithms and automata. When given a DIMACS CNF file, a SAT Solver might spend ages trying to find a solution that does not exist. As such, each DIMACS CNF file can be processed for a fixed period of time. If no solution is found within that time period, the result is *INDET* or *indeterminate*.

A number of open source SAT Solvers are available online. One of these is MiniSat [8], a minimalistic open source SAT Solver written in C++, developed to help researchers and developers alike to get started on SAT. MiniSat uses the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [9], which is a backtracking search algorithm that uses unit propagation and pure literal elimination. Unit propagation allows for clauses with only a single unassigned literal to be assigned greedily, which prunes the search space. Pure literal elimination happens when a literal only occurs in a single polarity, which allows all clauses containing it to be true, further pruning the search space. MiniSat has won numerous awards in SAT competitions. MiniSat is also branched as CryptoMiniSat. CryptoMiniSat is a multi-threaded SAT Solver released under a MIT open source license. Another notable branch of MiniSat is Glucose.

Another SAT Solver is Lingeling, by Armin Biere from the Johannes Kepler University at Linz [10]. Lingeling is implemented in C, and has an API. The base code of Lingeling has been used in a number of reimplementations, notably Plingeling, Clingeling, Splatz, and PicoSAT. Lingeling uses a number of preprocessing algorithms to

prune the search space, before feeding the problem to a Conflict-Driven Clause Learning (CDCL) algorithm [11], inspired by DPLL. The two differ mainly in CDCL backjumping non-chronologically; where DPLL backtracks by chronologically retracing arbitrary decisions, CDCL retraces arbitrary decisions which led to conflicts non-chronologically. It also adds these conflicts as inverted clauses to the problem.

FastSatSolver is a C++ SAT Solver which uses Genetic Algorithms to solve SAT problems. Using Genetic Algorithms allows it to rapidly solve some instances of SAT problems.

Chapter 8

Generating timetables

In order to test the strength and usability of using a SAT Solver to find minimal conflicting teacher sets in HSTT, we required a set of different example files. To facilitate this, a generator was created that is able to supply templates for schools of varying sizes. These generated timetables can then be used as problem sets.

The input for generating a timetable is an integer value between 0 and 20. This value is modified slightly to allow for variance.

Using the input value n, we add to the file in a fixed order:

1. **Departments** To correctly simulate Dutch high schools, we use the value to generate n departments with a prefix *M*, $n + 1$ departments with a prefix *H*, and $n + 2$ departments with prefix *V*.

2. **Students** Each department is populated with a number of students, which is $10 \times n$, divided by a random number r, where $0 < r < 4$. This number is then slightly tweaked for further randomization. Each student takes part in exactly one department. The students of each department are all assigned to basic classes, to which all courses of mandatory subjects are attached.

3. **Subjects** A fixed number of 27 common Dutch subjects is added to the schedule.

4. **Packages** Each student is given a set of subjects it will follow courses of. For each department, s subjects are mandatory, where $7 < s < 15$. In case of a mandatory subject each student will be assigned that subjects. Other subjects are optional, and not all students are assigned those subjects. Each student is supplied with a number of subjects in such a way that the total number of courses he or she would follow lies within acceptable bounds (between 19 and 35 courses).

5. **Groups** Students are assigned to groups. Groups will later on be used to assign to courses. For mandatory subjects, the basic classes of each department is used. For optional subjects, groups are generated, using a maximum number of students of 30.

6. **Courses** Once groups have been generated, we create courses for each combination of group and subject.

7. **Teachers** Teachers are generated with unique, random, three-letter user codes. Teachers are assigned an Full-Time Equivalent (FTE) between 0.2 and 1.0. Then, teachers are assigned all courses for a set of unique combinations of groups and subjects.

8. **Constraints** Teachers with an FTE below 1.0 are supplied with aspecific schedule constraints to limit the freedom of their solutions.

The students in the generated timetables are not assigned into groups for optional subjects. Assigning students into groups for optional subjects influences what courses can be scheduled at the same time. Assigning students in such a way that the department as a whole can be optimally scheduled is a separate problem altogether [12] and falls well outside the scope of this thesis. The software used to generate timetables also contains software used to assign the students into groups.

Chapter 9

Searching for Minimal Conflicting Teacher Sets

Once a scheduling problem is modelled in a DIMACS CNF file, an open source SAT Solver is able to check the satisfiability of that scheduling problem. If a scheduling problem is *unsatisfiable*, then we know that the modelled scheduling problem (all courses and the assignments of teachers and students to those courses) cannot be scheduled in such a way that all constraints imposed on it are satisfied.

A school cannot be run without a schedule, and so, it is important to find out which set of constraints is the possible cause of the unsatisfiable schedule. If a minimal set of conflicting constraints can be found, then the school scheduler knows what the cause of the unsatisfiability is, and can try to solve that problem, either by changing the constraints imposed on the schedule, or by reassigning teacher or students to different courses.

As decribed in Section 6.1, a scheduling problem cannot be modelled in a DIMACS CNF file without at least modelling L_m and C_m, where L_m is the set of all literals representing the scheduling of a course on a timeslot, and C_m is the set of clauses required to schedule each course only once and without conflicts. All other literals and clauses, contained in sets L_a and C_a, cater for other types of constraints.

We define a Conflicting Teacher set as a set of teachers with the property that scheduling all courses of the scheduling problem without conflicts is impossible when imposing only the constraints of that set of teachers onto the schedule. We define a Minimal Conflicting Teacher (MCT) set as a Conflicting Teacher set with the additional property that all subsets of that set are not Conflicting Teacher sets.

The search space for Minimal Conflicting Teacher sets can be modelled as a directed rooted tree with a root node containing the set of all teachers. Each edge leading away from a node to another leads to a node of which the teacher set is a strict subset of the parent node.

Each node that contains a Conflicting Teacher set is *unsatisfiable*. Sets containing only

subsets of an MCT set are always *satisfiable*, and this can be checked by a SAT Solver in polynomial time for smaller set sizes. Checking all subsets of a large set of teachers is not feasible in polynomial time. As such, identifying a node of which the teacher set is an MCT set requires a good search strategy to reduce running time. The teacher set of a node of size n can only be marked as an MCT set if its schedule is *unsatisfiable* and none of the schedules of its child nodes of size $n-1$ are *unsatisfiable*.

In this chapter we will describe two algorithms to find MCT sets. The first algorithm uses a bottom-up approach, while the second algorithm uses a top-down approach. Both algorithms return a set of nodes of which the teacher set of each node is an MCT set. Whenever a SAT Solver is used to check for a solution to the scheduling problem of a node, a file is written that contains all literals and clauses in L_m and C_m, and which contains only the clauses and literals in L_a and C_a used to describe the schedule demands of the teachers in that node's teacher set.

9.1 A first algorithm to find MCT sets

Using the basic operations of modelling a scheduling problem in a DIMACS CNF file, starting and stopping a SAT Solver, and reading the results of a SAT Solver run, we can create an algorithm that allows us to systematically search through search space to find MCT sets. The pseudocode for the first algorithm that was created can be seen in Algorithm 1. The algorithm is named FINDMCTSETSBOTTOMUP.

Algorithm 1 FINDMCTSETSBOTTOMUP(**set** T, **int** *maxRounds*, **int** *initialSetSize*)

```
int round ← 1
Tree x ← T
repeat
    if round < maxRounds or allNodesBranchedFromSingleNode() then
        removeSatisfiedNodes(x)
        createNodes(x, initialSetSize, T)
    else
        mergeSatisfiedNodes(x)
    end if
    branchUnsatisfiedNodes(x)
    runSATSolverOnNodes(x)
    checkSatisfiedBranchedNodesAndMarkParentBranchIfMinimal(x)
    round ← round + 1
until userInterrupts()
return getAllMarkedMinimalNodes(x)
```

Algorithm 1 searches for MCT sets within set T, which contains all user-selected teach-

ers in a scheduling problem. The algorithm starts in an exploratory phase lasting at most *maxRounds* rounds, in which it removes all *satisfied* nodes from the current tree x using *removeSatisfiedNodes*(x) and then creates new nodes populated randomly with teacher sets of size *initialSetSize* (using *createNodes*), which is user-supplied. It uses the SAT solver to check the solvability of each node, then marks them as either *satisfied* or *unsatisfied*. After the exploratory phase of the algorithm, nodes marked as *satisfied* are merged into other nodes marked as *satisfied* until a node is found which is marked as *unsatisfied*. If at any point during the run of the algorithm an *unsatisfied* node is found, then the algorithm checks all subsets of size $n-1$, where n is the *unsatisfied* parent node's teacher set size, an operation called *branching*. If all of those branched nodes contain teacher sets that are *satisfiable*, then the node is marked as containing a Minimal Conflicting Teacher set. *Unsatisfied* nodes are never removed from x. If all of the nodes in x are branched from a single node, then the algorithm creates new nodes to prevent itself from being unable to find new MCT sets. After the user interrupts the algorithm, the set of all nodes in x marked as an MCT set are returned by *getAllMarkedMinimalNodes*(x).

The operation of branching is a **breadth-first search strategy**. Breadth-first searches are executed by creating n child nodes from each *unsatisfied* node, where n is the number of teachers in the *unsatisfied* node. Each of the child nodes is populated with $n-1$ teachers, so that each subset of size $n-1$ is present. The scheduling problem of each of these child nodes is checked by the SAT Solver, which is costly, especially at larger sizes and with many unsatifiable sets. This strategy systematically explores the total search space and can guarantee that all MCT sets present in the parent node are found, at the cost of running time. Since the algorithm starts with many nodes containing small teacher sets, this algorithm uses a bottom-up approach, wherein it hopes to come across a Conflicting Teacher Set rapidly, while still able to check a solution for satisfiability rapidly. As teacher set sizes increase, so does the computational cost of searching for an MCT set within that teacher set.

During the exploration phase of this algorithm, it creates multiple nodes containing a small teacher set. These sets are later merged together if no *unsatisfied* teacher set is found. The algorithm is using a **bottom-up search strategy**, which can be preferable in scheduling problems containing a large number of small MCT sets. In a bottom-up search strategy, we create a number of child nodes such that their teacher sets are a partition of the total teacher set. These nodes are then scheduled by a SAT Solver. If any solution is *unsatisfied*, then a breadth-first search strategy can be used to find MCT sets from that solution.

This strategy heavily benefits when searching a scheduling problem containing small MCT sets, but suffers greatly when the smallest MCT sets are larger than the initial set size. It is quite common for this strategy to have to merge a small number of nodes with large teacher sets to find an MCT set, which negatively affects search time.

Once the user interrupts the algorithm, each unsatisfied node marked as an MCT set is returned.

9.2 A second algorithm to find MCT sets

We now introducte a second algorithm that searches for MCT sets, Algorithm 2. This algorithm uses a top-down approach, seeking to rapidly eliminate teachers from the root node containing all user-selected teachers. The algorithm is named FINDMCTSETSTOPDOWN.

Algorithm 2 FINDMCTSETSTOPDOWN(**set** T, **int** *upperBound*)

```
Tree x ← T                                  ▷ Tree with root node x, marked 'indeterminate'
repeat
    Tree y ← findSmallestUnsatisfiedNode(x)
    if y not null then
        if containsChildNodes(y) then
            if teacherSetSize(smallestChildNode(y)) = 1 then
                removeChildNodes(y)
                branchNode(y)
            else
                moveTeachersFromSmallestToLargestChildNode(y)
            end if
        else
            if teacherSetSize(y)/2 ≥ upperBound then
                splitNode(y)
            else
                branchNode(y)
            end if
        end if
    end if
    runSATSolverOnNodes(x)
    checkSatisfiedBranchedNodesAndMarkBranchIfMinimal(x)
until userInterrupts() or isSatisfiable(x)
return getAllMarkedMinimalNodes(x)
```

Algorithm 2 uses a **greedy depth-first search strategy** which aims at reducing the number of steps required to find an MCT set compared to depth-first searching. The algorithm first creates a root node called x. Each node consists of a teacher set, a marking, a set of pointers to children, and a status. The marking of a node can be *satisfied*, *unsatisfied*, or the default marking of *unknown*. The status can be *split*, *branched*, or *child*. Root node x's teacher set is initialized with all user-selected teachers, marked as *unknown* and with status *child*. It has no children.

The algorithm next recursively looks up the node with the smallest teacher set size marked as *unsatisfied* within the tree of root x and returns it as node y. If multiple nodes are found with the same teacher set size, the first encountered is returned.

If y is returned that node either contains only *satisfied* child nodes (since all children are smaller than their parents), or contains no children at all. If y contains no children, then it either branches when half its children rounded down is below a user-supplied upper bound, or it creates two child nodes through *splitNode*(y), an act called *splitting*. These child nodes contain teacher sets of size $\lceil n/2 \rceil$, resp. $\lfloor n/2 \rfloor$, where n is the number of teachers of its parent. These sets are marked as *unknown* and are a partition of the parent node.

An initial allocation of teachers to sets was implemented using completely randomly assigning of teachers to sets. This implementation was improved by implementing a distance matrix M, which contains the distances between any two teachers. The distance between two teachers is defined as the number of their courses that share a student, classroom, or teacher. The first teacher is added to the set randomly, with all subsequent teachers selected by sorting the previous teacher's distance values in M and selecting the teacher with the highest score. Some small amout of randomization is added to allow for some variance in selection of teachers.

If y does contain children, then, if the smallest child node of y contains a single teacher and y has status *split*, all child nodes of y are removed and its assigned status becomes *branched*. If the smallest child node of y contains multiple teachers, and y has status *split*, then teachers are moved from the child node with the smallest teacher set to the child node with the largest teacher set through *moveTeachersFromSmallestToLargestChildNode*(y). If both are of the same size, teachers are moved from the second child node. Teachers selected to be moved are selected at random.

Once teachers have been moved from one child node to another, both sets are marked as *unknown*, its initial marking, such that the algorithm retries solving both nodes with the SAT Solver.

Because greedy depth-first searching never checks all possible child nodes of size $n-1$, it alone cannot be used to find MCT sets. Instead, it applies a breadth-first search strategy if the size of the teacher set in its largest child node is below upper bound u.

If a SAT Solver cannot find a solution within the time period it is allotted, then the solution is labelled *indeterminate*. Solutions labelled *indeterminate* are considered to be *unsatisfiable* to the algorithm. If the root node x is *satisfiable*, then no MCT sets are present, and the algorithm exits.

Optimizing the search algorithm

There are a number of variables that can be adapted to modify the behaviour of Algorithm 2.

The total time limit the algorithm is allowed to run can be modified. This is useful when used in combination with different sets of initial parameters. Limiting the total time the algorithm is allowed to run enables it to restart with different parameters. Restarting with different parameters allows the algorithm to find solutions it would otherwise not have found, such as MCT sets of a size larger than the upper bound for branching.

The time period a SAT Solver is allowed to search for a solution can be modified. During the initial search, the set of teachers is large, and so it is advisable to keep the time a SAT Solver is allowed to search for solutions small, as it might not be able to find any solution in polynomial time. Child nodes however contain a reduced number of teachers, and with it a reduced number of literals the SAT Solver needs to check. Expanding the time the SAT Solver is allowed to spend on finding a solution for these smaller scheduling problems might allow for fewer *indeterminate* solutions, and harder proof for how conflicting a set of teachers is.

Another variable that can be tweaked is the upper bound u of a node, which dictates when a node starts to create child nodes using a breadth-first strategy. Using a lower value saves time, as fewer child nodes are generated, though generating schedules for all the subsets of size $n-1$ is still quite fast for values of n with $1 < n < 20$.

Different search strategies

Algorithm 2 is a depth-first greedy algorithm which seeks to rapidly find any minimal conflicting teacher set. It can however be changed to behave differently, which might make it more suitable to find minimal conflicting teacher sets in different scheduling problems.

One such strategy might be to start with a large number of nodes with subsets of teachers of size n with $5 < n < 10$. If the scheduling problem contains multiple pairs or triples of teachers that are conflicting, this might be a faster way to find these subsets, than to start with all teachers, and using a binary search strategy. It is also possible to construct nodes for each pair of teachers, or each triple. Assuming 100 teachers on average, this would yield about 5,000 unique pairs or about 160,000 unique triples.

Obviously, it is always useful to check the initial selection of all teachers, as finding a solution for the scheduling problem of all teachers means that there are no conflicting teacher sets at all.

9.3 Making Algorithm 2 more efficient

Algorithm 2 was adapted after running some computational experiments, in order to reduce running times for sets with large MCT subsets. The most notable change was to disallow the breadth-first search strategy in child nodes of size $n > 50$. If all child nodes of a node with a larger size are satisfiable, then binary searching is simply reset and the algorithm tries again. This does however mean that MCT sets of size > 50 would never be found by the algorithm. This is an acceptable sacrifice, as sets of size larger than 50 are exceedingly rare. It also means that sets of larger sizes will be harder to find, because binary searching might reset multiple times if conflicting teachers keep finding their way into the smaller child node. This algorithm was then used on both problem sets. In addition, it also kept record of the number of rounds it took to find a solution, as it is possible that finding a

solution with only binary searching still takes a long time if all unsatifiable sets were found in highly unbalanced child nodes.

Chapter 10

Computational experiments

In this chapter we describe seceral experiments aimed at finding MCT sets. In these experiments, we ran Algorithm 1 and both versions of Algorithm 2 searching for MCT sets, and stored information about running time, number of conversion steps, number of breadth-first searches, and the MCT sets found. Each of these runs was ended the moment a MCT set was found.

All experiments ran on a Dell Precision 3510 containing a 2.6-GHz Intel Core i7-6820HQ CPU with 16GB RAM memory running ElementaryOS 0.4 Loki, which is Ubuntu 16.04 based. All timetables were generated by proprietary software within a virtualized Windows 10 environment, compiled by Virtual Studio 2017 RC. MiniSat [8] was compiled using version 2.2.0 of its source code with GCC 5.4.0 [13].

We generated a number of test scheduling problems for the SAT Solver to solve. In each of these files students have been assigned to classes and constraints were added that were exportable to DIMACS CNF format. Each of these scheduling problems was converted to a DIMACS CNF file, after which MiniSat was used to search for a solution.

10.1 Problem Set 1

A first experiment was run using generated Problem Set 1. This Problem Set was selected because it could not be scheduled without conflicts by the automata of the software, and was of a comparable size to a typical Dutch high school. There were 113 teachers in the scheduling problem, all assigned a set of constraints limiting the total available timeslots to a value corresponding to their Full-Time Equivalent (FTE). All of these constraints were specific schedule constraints (i.e., fixed timeslots blocked in the schedule of the teacher). Before running the experiments, a different version of Algorithm 2 was run which did not stop when finding an MCT set, and which systematically explored all child nodes of all unsatisfied nodes until no unsatisfied child nodes could be found containing a teacher set which was not already known to be an MCT set. Four distinct MCT sets were found of

sizes 7, 10, 11, and 20.

Algorithm 1 was allowed to run without a timeout and with an ititial set size of 21. This initial set size was chosen to allow the program to find all known MCT sets in the Problem Set, while still retaining the ability of generating and testing multiple teacher sets during each round. The algorithm was allowed to work with 50 exploration rounds before merging the satisfied teacher sets. This value was chosen to ensure that enough exploration rounds were attempted before merging the sets, which could lead to a large unsatisfied set that would need to be branched: an expensive operation with regards to time.

Minimum Conflicting Teacher set (teacher codes)	Size	Found
kth, nrt, roz, ppq, wzo, zeb, vzn	7	44×
bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	56×

Table 10.1: MCT sets found by Algorithm 1 after 100 runs on Problem Set 1.

Both versions of Algorithm 2 were also allowed to run without a timeout and with a breadth-first lower bound of 8 until a first conflicting teacher set was found. The lower bound was set to prevent the algorithm from branching too early, which is expensive with regards to time, while still allowing the algorithm to find all MCT sets. Algorithm 2 found 4 distinct Minimal Conflicting Teacher sets:

Minimum Conflicting Teacher set (teacher codes)	Size	Found
kth, nrt, roz, ppq, wzo, zeb, vzn	7	20×
bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	58×
dvb, chm, heg, knc, msv, roo, pne, ngs, sbl, qry, zne	11	21×
hzr, elr, cwx, cqi, ftk, nku, mte, lur, ngs, jbu, tur, vlb, slg, pkf, stg, qfj, ppq, xnj, wuz, zsm	20	1×

Table 10.2: MCT sets found by Algorithm 2 after 100 runs on Problem Set 1.

Both algorithms found an MCT set during each run, and running times were such that a school scheduler for a Dutch high school could make use of the tool without having to wait too long. Table 10.3 shows for 100 runs of each algorithm for Problem Set 1 the total running times, the median running times, the standard deviation of the running times, and the number of MCT sets found.

Algorithm name	Total runtime	Median runtime	Standard deviation	nr. MCT sets found
Algorithm 2	1,684 minutes	17.8 minutes	701.17	4
Algorithm 2 (adapted)	933 minutes	8.22 minutes	23.96	3
Algorithm 1	1,873 minutes	10.16 minutes	11.98	2

Table 10.3: Statistics of 100 runs for Algorithms 1 and 2 on Problem Set 1.

The adapted version of Algorithm 2 clearly reduced the average running times for each MCT set found. The difference is most notable in larger sets, where a 48% decrease in running time was measured for the MCT set of size 11. Algorithm 1, though very consistent in running times, was also the poorest performer in running times and did not find the MCT sets of size 11 and 20. Figure 10.1 plots the running times of each algorithm. The consistent speed of Algorithm 1 is clearly visible, as is the improvement in running time between both versions of Algorithm 2.

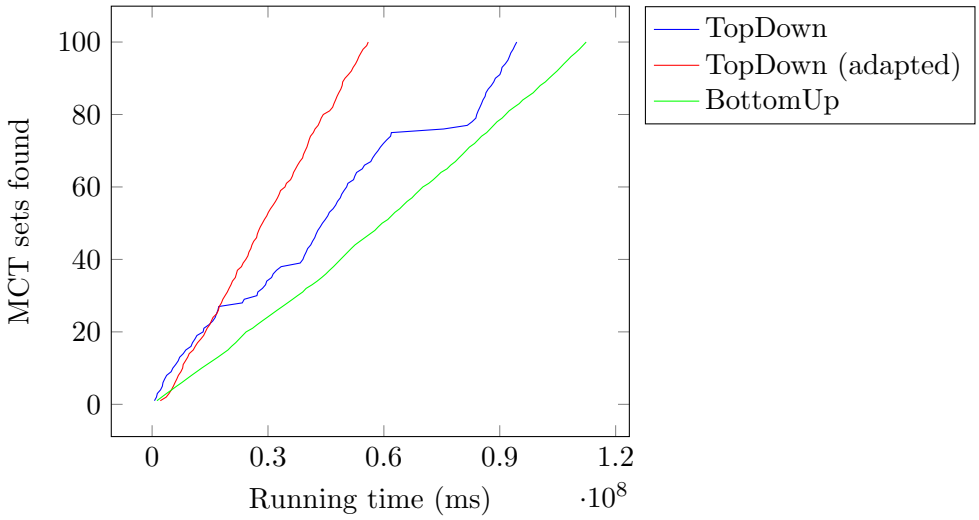

Figure 10.1: Running times of the algorithms for Problem Set 1.

10.2 Branching strategy

Table 10.4 illustrates the branching strategy at each point during a run which found an MCT set of size 11. Binary searches (B) only required two files to be converted and solved by a SAT Solver, while Breadth-first searches (W) required a file to be converted and solved by a SAT Solver for each teacher in the set, and were therefore computationally costly at larger set sizes. The root node of 113 teachers was split with a binary search strategy, which eventually found a conflicting teacher set of size 99 after merging half the teachers of the smaller set into the larger one twice. The first few Breadth-first searches at size 55 took a minimum of 10 minutes and reduced the size of the teacher set by 1, while the first Binary search took no more than one minute and reduced the teacher set size by 14. This clearly illustrates that using breadth-first searching on nodes with large teacher sets has a high computational cost, which is further shown by the notable difference in running times between both versions of Algorithm 2.

113	99	86	83	72	63	59	55	54	53	52	48	47	44	41	40	38	37	36
B	B	B	B	B	B	B	W	W	W	B	W	B	B	W	B	W	W	B
33	32	31	30	28	26	23	22	20	19	18	17	16	15	14	13	12	11	
W	W	W	B	B	B	W	B	W	W	W	W	W	W	W	W	W	W	

Table 10.4: Branching Strategy during a specific run of Algorithm 2 on Problem Set 1. B = Binary search strategy, W = Breadth-first search strategy

10.3 Problem Set 2

A second experiment was run using generated Problem Set 2. This Problem Set was selected because it *could* be scheduled without conflicts using traditional methods. It was then modified to contain exactly two MCT sets, both of size 2. The size of the scheduling problem was also comparable to that of a typical Dutch high school. There were 155 teachers in the scheduling problem, all assigned a set of constraints limiting the total available timeslots to a value corresponding to their Full-Time Equivalent (FTE). All of these constraints were specific schedule constraints (i.e., fixed timeslots blocked in the schedule of the teacher).

Algorithm 1 was allowed to run without a timeout and with an ititial set size of 5. This initial set size was lower than that of Problem Set 1 because of the smaller MCT set sizes in the Problem Set. The algorithm was allowed to work with 50 exploration rounds before merging the satisfied teacher sets.

Minimum Conflicting Teacher set (teacher codes)	Size	Found
uhu, upf	2	42×
xga, xma	2	58×

Table 10.5: MCT sets found by Algorithm 1 after 100 runs on Problem Set 2.

Both versions of Algorithm 2 were also allowed to run without a timeout and with a breadth-first lower bound of 8 until a first conflicting teacher set was found. The lower bound remained unchanged from Problem Set 1 because the effect of reducing running time is negligible at such low values.

Minimum Conflicting Teacher set (teacher codes)	Size	Found
uhu, upf	2	88×
xga, xma	2	112×

Table 10.6: Combined MCT sets found by both versions of Algorithm 2 after 100 runs on Problem Set 2 each.

Both algorithms found an MCT set during each run, and running times were much lower than the running times for Problem Set 1, which contained larger MCT sets. Table 10.7 shows for 100 runs of each algorithm for Problem Set 2 the total running times, the median running times, the standard deviation of the running times, and the number of MCT sets found.

Algorithm name	Total runtime	Median runtime	Standard deviation	nr. MCT sets found
Algorithm 2	97.84 minutes	0.94 minutes	0.96	2
Algorithm 2 (adapted)	99.92 minutes	0.97 minutes	1.01	2
Algorithm 1	65.13 minutes	0.49 minutes	0.08	2

Table 10.7: Statistics of 100 runs for Algorithms 1 and 2 on Problem Set 2.

Running times for this Problem Set are much reduced, as are the standard deviations for each algorithm. All algorithms found both MCT sets present in the Problem Set.

Figure 10.2 shows a plot of running times for each algorithm for 100 runs on Problem Set 2. Though both versions of Algorithm 2 rapidly manage to reduce the initial teacher set size from 155 to a MCT set of size 2, Algorithm 1 succeeds in finding either of the two MCT sets much more rapidly by populating many small sets and testing them. MCT sets of size 2 are somewhat contrived; in most cases, the scheduler would easily be able to pick these sets out straightforwardly, or analyze the problem at such pace that searching for them algorithmically does not benefit much. The difference in runtime between both versions of Algorithm 2 is less than 3% and appears to be insignificant.

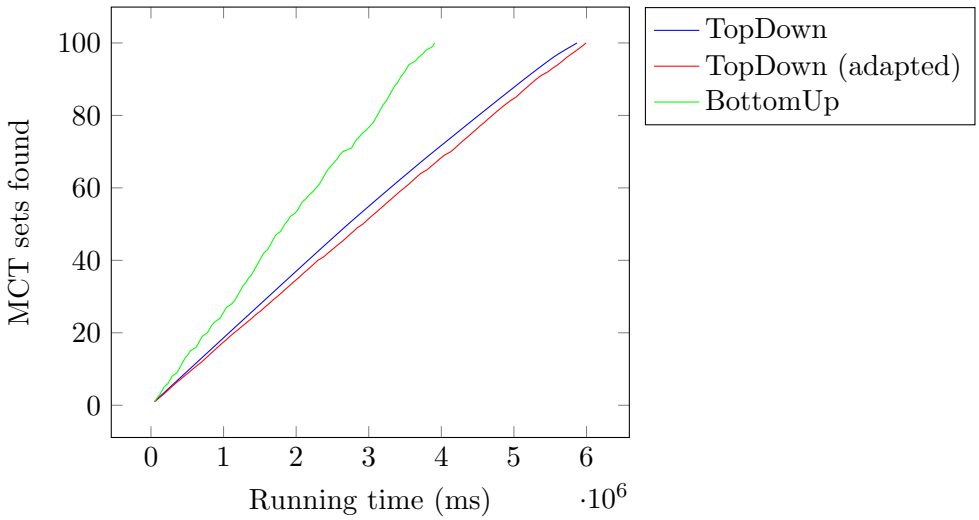

Figure 10.2: Running times of the algorithms for Problem Set 2.

10.4 Performance of both versions of Algorithm 2 on Problem Set 1

The running times for each run of both versions of Algorithm 2 are strongly connected to the number of intermediate steps required to reduce the current conflicting teacher set of a node to an MCT set. Figure 10.3 plots the running time of each run of Algorithm 2 on Problem Set 1 against the total number of intermediate steps in which an unsatisfiable teacher set was found. Note that the horizontal axis has a logarithmic scale.

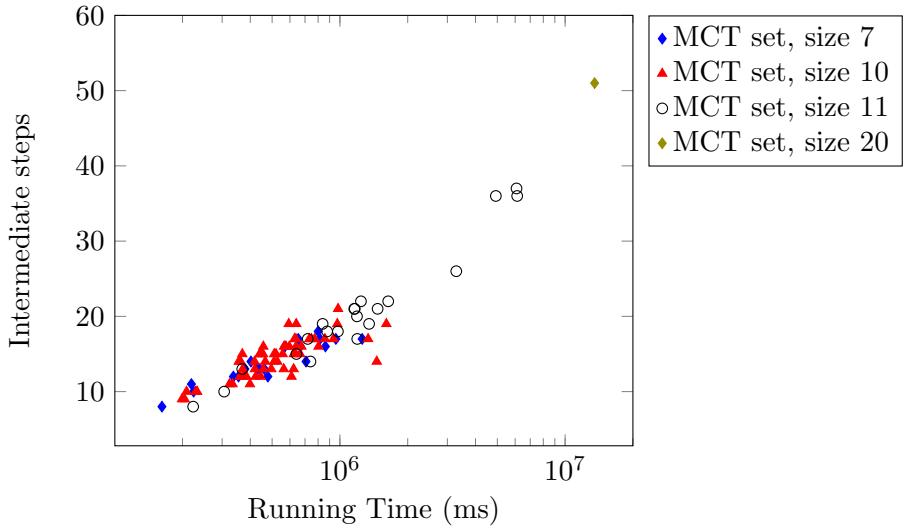

Figure 10.3: Running times for Algorithm 2 for 100 runs on Problem Set 1. Note that the horizontal axis has a logarithmic scale.

Figure 10.3 clearly shows that smaller MCT sets require on average fewer intermediate steps than larger MCT sets. Figure 10.4 shows the same plot for 100 runs of the adapted version of Algorithm 2 on Problem Set 1.

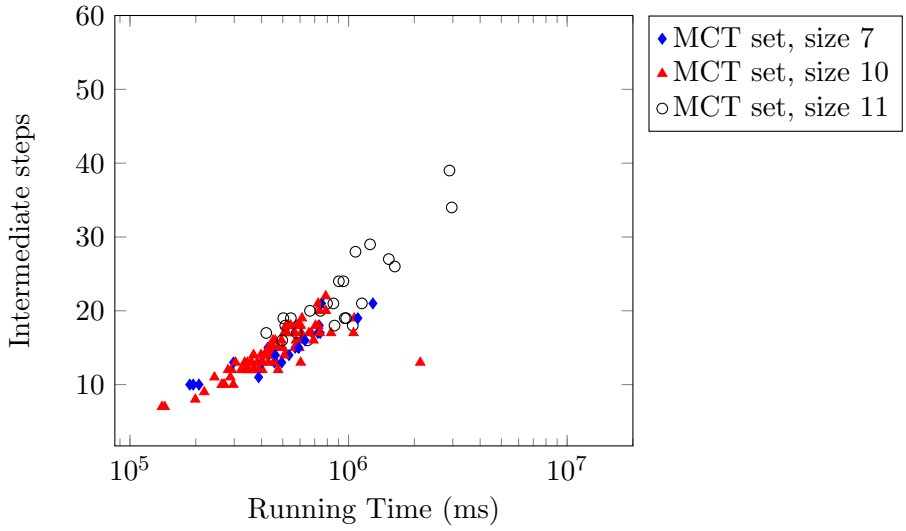

Figure 10.4: Running times for the adapted version of Algorithm 2 for 100 runs on Problem Set 1. Note that the horizontal axis has a logarithmic scale.

Figure 10.4 reveals that the number of intermediate steps required by the adapted version of Algorithm 2 are less varied for each MCT set it found. The running time is also lower for larger MCT set sizes.

10.5 Performance of both versions of Algorithm 2 on Problem Set 2

Both versions of Algorithm 2 showed comparable performance in Problem Set 2, with regards to total number of conversion steps and total running time. Both algorithms also found the individual MCT sets in roughly the same ratio. The reduced total runtime and number of intermediate steps found between both versions of Algorithm 2 in Problem Set 1 are not visible in this Problem Set. This is not unexpected; the key change between the original algorithm and the adapted version is the inclusion of a lower bound for *branching*. Because Algorithm 2 is able to rapidly reduce its current conflicting teacher set in the first few rounds of a run for problem sets with small MCT sets, the algorithm doesn't greatly vary in results beyond what is to be expected from random variance.

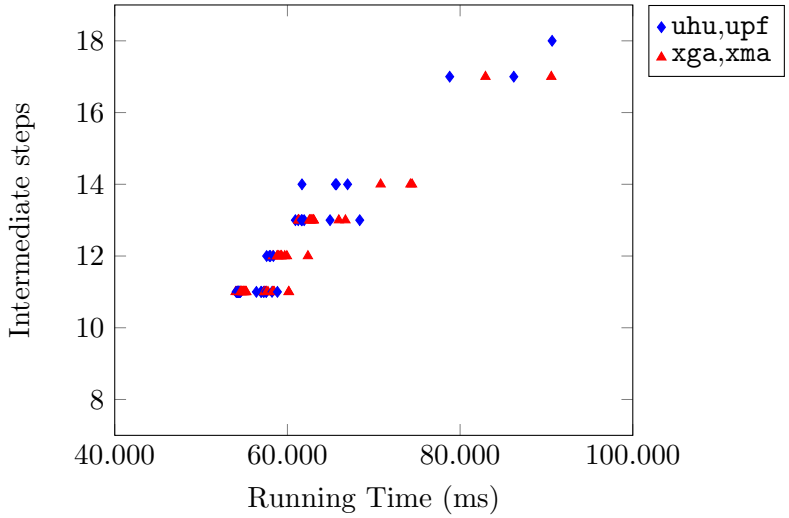

Figure 10.5: Running times for Algorithm 2 for 100 runs on Problem Set 2.

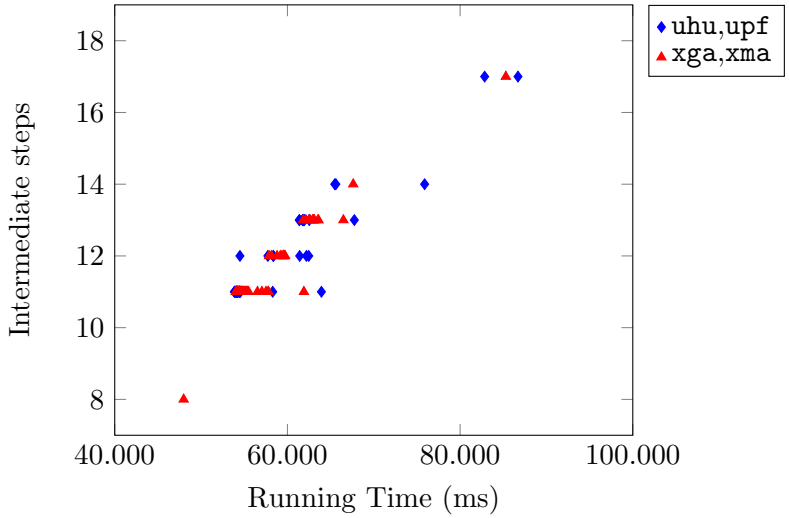

Figure 10.6: Running times for the adapted version of Algorithm 2 for 100 runs on Problem Set 2.

In short, both Algorithm 1 and Algorithm 2 are useful in finding MCT sets for various Problem Sets. Algorithm 1's approach of generating large numbers of small teacher sets to be tested allows it to rapidly find smaller MCT sets, but ultimately proves inefficient

when looking in Problem Sets containing only large MCT sets. Algorithm 2 benefits from running being able to rapidly reduce the total teacher set to a smaller set containing an MCT set, but is less efficient in Problem Sets containing small MCT sets.

The adapted version of Algorithm 2 shows a clear performance increase in Problem Sets containing large MCT sets, but cannot find sets of size larger than or equal to 50 due to its upper limit for branching strategies. This sacrifice appears to be minor, as such large MCT sets are rare. If a suspicion exists that such a large MCT set is present in a Problem Set, the algorithm can always be run with a higher upper limit.

Chapter 11

Conclusions and further research

In this thesis, we have tried to develop a strategy for rapid discovery of Minimal Conflicting Teacher (MCT) sets within instances of the Dutch High School Time Tabling problem. Though a simple heuristic can be used in small problems to locate these Minimal Conflicting Teacher sets, these heuristics perform poorly in problems of the size of a typical Dutch high school.

Using a binary searching algorithm drastically reduces the time required to discover a MCT set if the set is small, compared to breadth-first searching heuristics. Larger minimal conflicting teacher set sizes profit less from this approach, though smart gathering of teachers in sets can lead to faster MCT set discovery. One notable possibility of smart population of teacher sets is by using the distance matrix to populate the smallest set with teachers that are the least-connected.

Another algorithm, which uses a strategy of testing a large number teacher sets of small size, outperforms a binary searching strategy in Problem Sets containing minimal conflicting teacher sets of small size.

Currently, finding minimal conflicting teacher sets in scheduling problems of Dutch high schools depended on analysis performed by the school scheduler, and were ineffective for larger minimal conflicting teacher set sizes. Using these algorithms allows the school scheduler to identify minimal conflicting teacher sets, and do so at rapid pace. The ability to identify such a set will allow school schedulers to identify and implement solutions to such problems, such as changing teacher-to-class assignments, or by changing course or schedule constraints at a moment when they are not set in stone yet.

Further research is required to expand the number of constraints checked by the SAT Solver. Notably, the ability to check overflow on schedule timeslots for specified sets of courses is an important improvement and should allow for the discovery of many more MCT sets. If the SAT Solver is able to generate solutions wherin students are reassigned into different classes, more solutions will be possible, which should reduce the number of MCT sets.

Each of the algorithms uses a number of preset variables that can be tweaked to modify its behaviour. Though an effective set of these variables has been used, no in-depth study has been made into the effects of different values for each of these variables on the performance of the algorithms.

Glossary

course A course is a meeting between teachers and students in a classroom, which repeats for each timeslot it is scheduled on.

gap When a student or teacher's schedule contains a number of available timeslots directly preceded and succeeded by an unavailable timeslot, those timeslots are gaps.

period A timeslot is a time period with a fixed start and end time, which repeats for every day of a schedule. In most Dutch High Schools, all periods are of identical length, between 40 and 70 minutes.

schedule A schedule is set of courses, each scheduled on a timeslot, a day, a segment, and containing sets of students, teachers, and classrooms. Each course is scheduled such that no student, teacher, or classroom is required to take part in multiple courses at the same time.

segment A schedule can be partitioned into segments, such as trimesters or semesters.

student Students are taught subjects during their education at a High School. Students select a number of subjects from all subjects available to them to follow courses in during the schedule of a schoolyear.

subject A course contains a subject, which is what is taught by the teachers to the students.

timeslot A timeslot is a combination of a workday, a period, and a segment. If a student or teacher has no course scheduled for that timeslot, it is considered available, otherwise it is unavailable.

Appendix A

Results of running Algorithm 1

Algorithm 1 has been run on two problem sets.

A.1 Results for Problem Set 1

Run	Set	Set size	Time ms	Time min
1	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1261640	21.03
2	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1230563	20.51
3	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1253250	20.89
4	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1327344	22.12
5	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1261843	21.03
6	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1350782	22.51
7	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1257609	20.96
8	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1268484	21.14
9	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1317407	21.96
10	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1302218	21.70
11	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1390875	23.18
12	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1395375	23.26
13	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1393344	23.22
14	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1311797	21.86
15	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1310140	21.84
16	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	927969	15.47
17	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1067937	17.80
18	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	899969	15.00
19	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	906797	15.11
20	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	907062	15.12
21	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1663172	27.72
22	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1200578	20.01
23	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1275672	21.26
24	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1321438	22.02
25	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1298015	21.63
26	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1295016	21.58
27	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1344625	22.41
28	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1323672	22.06
29	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1306406	21.77
30	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1278266	21.30
31	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1338843	22.31
32	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	885032	14.75
33	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1529593	25.49
34	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1320235	22.00
35	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1176640	19.61
36	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1090375	18.17
37	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	941516	15.69
38	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1019859	17.00
39	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	919141	15.32
40	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	910641	15.18
41	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	922718	15.38

42	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	923079	15.38
43	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	922656	15.38
44	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	969406	16.16
45	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1269922	21.17
46	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1284969	21.42
47	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1287000	21.45
48	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1304031	21.73
49	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	890734	14.85
50	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	989375	16.49
51	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1454500	24.24
52	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	932657	15.54
53	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	928828	15.48
54	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1252062	20.87
55	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	927000	15.45
56	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	967266	16.12
57	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1268766	21.15
58	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	881437	14.69
59	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	981547	16.36
60	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	945484	15.76
61	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1537109	25.62
62	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1161266	19.35
63	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1013437	16.89
64	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	961610	16.03
65	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1584281	26.40
66	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	933625	15.56
67	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1230000	20.50
68	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	893687	14.89
69	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	960985	16.02
70	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	903734	15.06
71	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	880984	14.68
72	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1325516	22.09
73	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	914265	15.24
74	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	893516	14.89
75	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1332422	22.21
76	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	906750	15.11
77	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	892062	14.87
78	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	888375	14.81
79	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1301688	21.69
80	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	924219	15.40
81	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	889375	14.82
82	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1283203	21.39
83	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1341312	22.36
84	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	902875	15.05
85	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1371141	22.85
86	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1345422	22.42
87	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	906531	15.11
88	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	914109	15.24
89	kth, nrt, ppq, roz, vzn, wzo, zeb	7	1330125	22.17
90	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	959032	15.98
91	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1015844	16.93
92	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1042594	17.38
93	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	904016	15.07
94	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	884875	14.75
95	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	948156	15.80
96	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	896969	14.95
97	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1125031	18.75
98	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	1015016	16.92
99	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	890969	14.85
100	air, bfo, bov, glk, kth, nrt, ppq, roz, vzn, zeb	10	878015	14.63

Table A.1: Results of running Algorithm 1 on Problem Set 1.

A.2 Results for Problem Set 2

Run	Set	Set size	Time ms	Time min
1	xga, xma	2	55578	0.93

2	xga, xma	2	27313	0.46
3	uhu, upf	2	41297	0.69
4	xga, xma	2	27625	0.46
5	xga, xma	2	27125	0.45
6	uhu, upf	2	55672	0.93
7	uhu, upf	2	27218	0.45
8	xga, xma	2	27625	0.46
9	uhu, upf	2	69782	1.16
10	xga, xma	2	27562	0.46
11	xga, xma	2	27750	0.46
12	uhu, upf	2	26953	0.45
13	uhu, upf	2	27188	0.45
14	uhu, upf	2	42828	0.71
15	uhu, upf	2	27375	0.46
16	xga, xma	2	84500	1.41
17	uhu, upf	2	27328	0.46
18	uhu, upf	2	27203	0.45
19	xga, xma	2	27469	0.46
20	xga, xma	2	72063	1.20
21	xga, xma	2	29547	0.49
22	uhu, upf	2	27281	0.45
23	xga, xma	2	44562	0.74
24	xga, xma	2	74235	1.24
25	uhu, upf	2	27125	0.45
26	uhu, upf	2	29015	0.48
27	uhu, upf	2	29391	0.49
28	xga, xma	2	74547	1.24
29	xga, xma	2	42937	0.72
30	xga, xma	2	29328	0.49
31	xga, xma	2	27172	0.45
32	xga, xma	2	29032	0.48
33	xga, xma	2	27390	0.46
34	xga, xma	2	44344	0.74
35	xga, xma	2	27328	0.46
36	xga, xma	2	43609	0.73
37	uhu, upf	2	28297	0.47
38	xga, xma	2	28407	0.47
39	xga, xma	2	27578	0.46
40	xga, xma	2	29203	0.49
41	xga, xma	2	27578	0.46
42	xga, xma	2	28359	0.47
43	xga, xma	2	55532	0.93
44	uhu, upf	2	27343	0.46
45	xga, xma	2	27375	0.46
46	uhu, upf	2	27172	0.45
47	xga, xma	2	27344	0.46
48	uhu, upf	2	69156	1.15
49	xga, xma	2	27360	0.46
50	xga, xma	2	29500	0.49
51	uhu, upf	2	41765	0.70
52	uhu, upf	2	27422	0.46
53	uhu, upf	2	69547	1.16
54	xga, xma	2	41687	0.69
55	uhu, upf	2	27329	0.46
56	xga, xma	2	27640	0.46
57	uhu, upf	2	55766	0.93
58	uhu, upf	2	41547	0.69
59	xga, xma	2	55484	0.92
60	uhu, upf	2	41531	0.69
61	uhu, upf	2	44485	0.74
62	xga, xma	2	29859	0.50
63	uhu, upf	2	27234	0.45
64	xga, xma	2	30047	0.50
65	uhu, upf	2	29063	0.48
66	xga, xma	2	43360	0.72
67	xga, xma	2	44859	0.75
68	xga, xma	2	43625	0.73
69	uhu, upf	2	30360	0.51
70	xga, xma	2	45578	0.76
71	xga, xma	2	121547	2.03
72	uhu, upf	2	27203	0.45
73	uhu, upf	2	29468	0.49
74	uhu, upf	2	46188	0.77

75	xga, xma	2	43203	0.72
76	xga, xma	2	58922	0.98
77	xga, xma	2	45766	0.76
78	xga, xma	2	45234	0.75
79	xga, xma	2	29562	0.49
80	xga, xma	2	27579	0.46
81	uhu, upf	2	28250	0.47
82	uhu, upf	2	27421	0.46
83	uhu, upf	2	27610	0.46
84	xga, xma	2	41687	0.69
85	uhu, upf	2	27485	0.46
86	uhu, upf	2	27406	0.46
87	xga, xma	2	27703	0.46
88	uhu, upf	2	27625	0.46
89	xga, xma	2	42734	0.71
90	xga, xma	2	28204	0.47
91	xga, xma	2	29765	0.50
92	xga, xma	2	42969	0.72
93	uhu, upf	2	28078	0.47
94	uhu, upf	2	28281	0.47
95	xga, xma	2	88313	1.47
96	uhu, upf	2	44219	0.74
97	xga, xma	2	60016	1.00
98	xga, xma	2	43422	0.72
99	xga, xma	2	90515	1.51
100	uhu, upf	2	28719	0.48

Table A.2: Results of running Algorithm 1 on Problem Set 2.

Appendix B

Results of running Algorithm 2

Algorithm 2 has been run on two problem sets. Both problem sets contained at least two unique minimal conflicting teacher sets.

B.1 Results for Problem Set 1

Run	Set	Set size	Time ms	Time min
1	kth, nrt, roz, ppq, wzo, zeb, vzn	7	162125	2.7
2	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	199047	3.32
3	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	204437	3.41
4	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	208500	3.48
5	roz, zeb, vzn, wzo, kth, nrt, ppq	7	219094	3.65
6	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	223047	3.72
7	nrt, kth, roz, vzn, ppq, zeb, wzo	7	224219	3.74
8	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	229188	3.82
9	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	232188	3.87
10	chm, dvb, heg, pne, knc, msv, ngs, sbl, roo, qry, zne	11	306266	5.1
11	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	323406	5.39
12	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	333703	5.56
13	kth, nrt, ppq, roz, vzn, zeb, wzo	7	336547	5.61
14	kth, nrt, roz, ppq, wzo, zeb, vzn	7	354672	5.91
15	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	355562	5.93
16	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	359656	5.99
17	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	362625	6.04
18	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	368375	6.14
19	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	368735	6.15
20	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	369984	6.17
21	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	370547	6.18
22	kth, nrt, roz, ppq, wzo, zeb, vzn	7	378782	6.31
23	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	384734	6.41
24	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	398828	6.65
25	kth, nrt, roz, ppq, wzo, zeb, vzn	7	402672	6.71
26	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	417859	6.96
27	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	419265	6.99
28	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	422656	7.04
29	roz, zeb, vzn, wzo, kth, nrt, ppq	7	428360	7.14
30	roz, zeb, vzn, wzo, kth, nrt, ppq	7	440406	7.34
31	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	440453	7.34
32	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	441985	7.37
33	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	444860	7.41
34	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	448172	7.47
35	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	455437	7.59
36	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	457203	7.62
37	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	457437	7.62
38	kth, nrt, ppq, roz, vzn, zeb, wzo	7	461875	7.7
39	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	466235	7.77

40	nrt, kth, roz, vzn, ppq, zeb, wzo	7	478828	7.98
41	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	495938	8.27
42	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	510391	8.51
43	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	511125	8.52
44	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	520390	8.67
45	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	527172	8.79
46	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	556797	9.28
47	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	563313	9.39
48	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	565438	9.42
49	kth, nrt, roz, ppq, wzo, zeb, vzn	7	570016	9.5
50	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	575781	9.6
51	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	592469	9.87
52	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	596031	9.93
53	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	609656	10.16
54	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	623172	10.39
55	glk, air, bfo, bov, kth, nrt, ppq, roz, vzn, zeb	10	624391	10.41
56	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	628938	10.48
57	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	636422	10.61
58	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	637734	10.63
59	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	638485	10.64
60	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	639891	10.66
61	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	640968	10.68
62	roz, zeb, vzn, wzo, kth, nrt, ppq	7	653375	10.89
63	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	665531	11.09
64	kth, nrt, roz, ppq, wzo, zeb, vzn	7	667641	11.13
65	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	674421	11.24
66	roz, zeb, vzn, wzo, kth, nrt, ppq	7	707125	11.79
67	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	719735	12
68	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	731562	12.19
69	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	739875	12.33
70	air, glk, bfo, bov, nrt, kth, roz, vzn, ppq, zeb	10	748031	12.47
71	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	773156	12.89
72	kth, nrt, roz, ppq, wzo, zeb, vzn	7	800844	13.35
73	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	803750	13.4
74	nrt, kth, roz, vzn, ppq, zeb, wzo	7	816437	13.61
75	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	838047	13.97
76	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	858125	14.3
77	roz, zeb, vzn, wzo, kth, nrt, ppq	7	862391	14.37
78	dvb, chm, heg, knc, msv, roo, pne, ngs, sbl, qry, zne	11	877766	14.63
79	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	936422	15.61
80	roz, zeb, vzn, wzo, kth, nrt, ppq	7	959594	15.99
81	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	973859	16.23
82	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	979547	16.33
83	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	981578	16.36
84	dvb, chm, heg, knc, msv, roo, pne, ngs, sbl, qry, zne	11	1158468	19.31
85	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	1163594	19.39
86	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	1190578	19.84
87	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	1194094	19.9
88	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	1239875	20.66
89	kth, nrt, ppq, roz, vzn, zeb, wzo	7	1255500	20.93
90	bov, bfo, air, glk, kth, nrt, roz, ppq, zeb, vzn	10	1334172	22.24
91	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	1347703	22.46
92	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	1457359	24.29
93	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	1470609	24.51
94	roz, zeb, vzn, bfo, bov, glk, air, kth, nrt, ppq	10	1607609	26.79
95	dvb, chm, heg, msv, knc, ngs, qry, sbl, pne, roo, zne	11	1638609	27.31
96	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	3286281	54.77
97	roo, sbl, zne, dvb, heg, chm, ngs, msv, knc, pne, qry	11	4925219	82.09
98	dvb, chm, heg, knc, msv, roo, pne, ngs, sbl, qry, zne	11	6084500	101.41
99	dvb, chm, heg, knc, msv, roo, pne, ngs, sbl, qry, zne	11	6125250	102.09
100	hzr, elr, cwx, cqi, ftk, nku, mte, lur, ngs, jbu, tur, vlb, slg, pkf, stg, qfj, ppq, xnj, wuz, zsm	20	13521281	225.35

Table B.1: Results of running Algorithm 2 on Problem Set 1.

B.2 Results for Problem Set 2

Running times for Problem Set 2 were much more uniform, because the MCT sets in the Problem Set only consisted of two teachers.

Run	Set	Set size	Time ms	Time min
1	xga,xma	2	47984	0.80
2	upf,uhu	2	53859	0.90
3	upf,uhu	2	53875	0.90
4	upf,uhu	2	53922	0.90
5	upf,uhu	2	53953	0.90
6	xga,xma	2	54016	0.90
7	upf,uhu	2	54031	0.90
8	xga,xma	2	54062	0.90
9	upf,uhu	2	54063	0.90
10	xga,xma	2	54078	0.90
11	upf,uhu	2	54094	0.90
12	upf,uhu	2	54094	0.90
13	upf,uhu	2	54125	0.90
14	xga,xma	2	54156	0.90
15	xga,xma	2	54156	0.90
16	upf,uhu	2	54157	0.90
17	upf,uhu	2	54172	0.90
18	upf,uhu	2	54172	0.90
19	upf,uhu	2	54188	0.90
20	upf,uhu	2	54235	0.90
21	xga,xma	2	54250	0.90
22	xga,xma	2	54266	0.90
23	xga,xma	2	54328	0.91
24	upf,uhu	2	54343	0.91
25	xga,xma	2	54390	0.91
26	xga,xma	2	54391	0.91
27	xga,xma	2	54406	0.91
28	xga,xma	2	54406	0.91
29	xga,xma	2	54437	0.91
30	upf,uhu	2	54453	0.91
31	xga,xma	2	54453	0.91
32	xga,xma	2	54469	0.91
33	upf,uhu	2	54469	0.91
34	upf,uhu	2	54500	0.91
35	upf,uhu	2	54515	0.91
36	upf,xga	2	54547	0.91
37	xga,xma	2	54578	0.91
38	xga,xma	2	54641	0.91
39	xga,xma	2	54672	0.91
40	xga,xma	2	54750	0.91
41	xma,xga	2	54750	0.91
42	xga,xma	2	54875	0.91
43	xga,xma	2	54954	0.92
44	xma,xga	2	55062	0.92
45	xga,xma	2	55125	0.92
46	xga,xma	2	55281	0.92
47	xga,xma	2	55312	0.92
48	xga,xma	2	55328	0.92
49	xga,xma	2	55500	0.93
50	xma,xga	2	56532	0.94
51	xga,xma	2	57047	0.95
52	xga,xma	2	57516	0.96
53	upf,uhu	2	57734	0.96
54	xga,xma	2	57766	0.96
55	xga,xma	2	57797	0.96
56	xga,xma	2	57875	0.96
57	xga,xma	2	58141	0.97
58	upf,uhu	2	58297	0.97
59	upf,uhu	2	58359	0.97
60	uhu,upf	2	58375	0.97
61	xga,xma	2	58797	0.98
62	xga,xma	2	59172	0.99
63	xga,xma	2	59375	0.99
64	xga,xma	2	59531	0.99

65	xma,xga	2	59562	0.99
66	xga,xma	2	59594	0.99
67	xga,xma	2	59625	0.99
68	xga,xma	2	59735	1.00
69	upf,uhu	2	61375	1.02
70	upf,uhu	2	61390	1.02
71	upf,uhu	2	61422	1.02
72	upf,uhu	2	61750	1.03
73	upf,uhu	2	61766	1.03
74	xga,xma	2	61828	1.03
75	upf,uhu	2	61875	1.03
76	upf,uhu	2	61890	1.03
77	xga,xma	2	61907	1.03
78	upf,uhu	2	61968	1.03
79	xma,xga	2	62141	1.04
80	upf,uhu	2	62203	1.04
81	upf,uhu	2	62469	1.04
82	xga,xma	2	62515	1.04
83	upf,uhu	2	62532	1.04
84	xma,xga	2	62625	1.04
85	xga,xma	2	62906	1.05
86	xga,xma	2	63000	1.05
87	xga,xma	2	63094	1.05
88	xma,xga	2	63094	1.05
89	xga,xma	2	63546	1.06
90	xga,xma	2	63593	1.06
91	upf,uhu	2	63938	1.07
92	upf,uhu	2	65484	1.09
93	uhu,upf	2	65609	1.09
94	xga,xma	2	66485	1.11
95	xga,xma	2	67625	1.13
96	upf,uhu	2	67750	1.13
97	upf,uhu	2	75891	1.26
98	upf,uhu	2	82844	1.38
99	xga,xma	2	85296	1.42
100	upf,uhu	2	86704	1.45

Table B.2: Results of running Algorithm 2 on Problem Set 2.

Appendix C

Results of running the adapted Algorithm 2

The adapted Algorithm 2 has been run on the same two problem sets as Algorithm 2.

C.1 Results for Problem Set 1

Run	Set	Set size	Time ms	Conversion steps
1	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	2.128.938	27
2	chm, dvb, qry, roo, sbl, heg, msv, ngs, knc, pne, zne	11	1.528.140	101
3	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	734.985	57
4	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	608.938	43
5	chm, dvb, qry, roo, sbl, heg, msv, ngs, knc, pne, zne	11	503.109	54
6	roz, wzo, vzn, kth, ppq, nrt, zeb	7	429.125	23
7	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	468.203	28
8	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	402.938	21
9	chm, dvb, qry, roo, sbl, heg, msv, ngs, knc, pne, zne	11	649.156	42
10	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	452.344	30
11	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	144.078	9
12	roz, wzo, vzn, kth, ppq, nrt, zeb	7	589.391	24
13	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	568.390	39
14	roz, wzo, vzn, kth, ppq, nrt, zeb	7	400.985	21
15	chm, dvb, qry, roo, sbl, heg, msv, ngs, knc, pne, zne	11	1.045.063	54
16	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	513.000	29
17	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	594.828	35
18	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	831.875	48
19	roz, wzo, vzn, kth, ppq, nrt, zeb	7	752.015	45
20	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	448.282	27
21	roz, wzo, vzn, kth, ppq, nrt, zeb	7	458.500	18
22	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	676.641	46
23	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	405.438	24
24	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	438.609	29
25	chm, dvb, qry, roo, sbl, heg, msv, ngs, knc, pne, zne	11	960.812	56
26	roz, wzo, vzn, kth, ppq, nrt, zeb	7	427.672	26
27	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	218.562	13
28	roz, wzo, vzn, kth, ppq, nrt, zeb	7	569.797	24
29	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	365.562	21
30	glk, bov, air, bfo, roz, vzn, kth, ppq, nrt, zeb	10	573.578	50
31	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	582.282	43
32	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	449.062	35
33	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	493.125	36
34	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	386.844	20
35	nrt, kth, ppq, roz, wzo, vzn, zeb	7	741.921	32
36	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	270.563	17

37	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	266.906	17
38	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	1.053.703	40
39	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	430.875	25
40	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	691.500	39
41	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	541.250	43
42	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	263.000	20
43	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	500.125	32
44	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	287.765	21
45	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	451.594	39
46	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	743.281	64
47	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	199.000	11
48	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	359.547	27
49	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	422.891	29
50	nrt, kth, ppq, roz, wzo, vzn, zeb	7	533.734	20
51	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	508.844	29
52	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	607.860	42
53	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	420.110	40
54	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	567.875	33
55	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	665.922	56
56	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	612.406	48
57	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	526.609	42
58	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	433.609	29
59	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	345.922	23
60	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	1.150.547	71
61	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	371.797	22
62	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	1.075.797	99
63	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	472.375	31
64	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	296.782	20
65	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	528.703	43
66	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	506.984	41
67	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	575.204	37
68	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	725.609	60
69	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	243.453	19
70	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	433.360	36
71	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	447.968	27
72	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	291.891	23
73	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	281.000	25
74	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	349.750	20
75	nrt, kth, ppq, roz, wzo, vzn, zeb	7	734.531	36
76	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	443.500	31
77	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	797.063	72
78	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	428.891	28
79	air, bfo, bov, nrt, kth, ppq, roz, glk, vzn, zeb	10	371.328	20
80	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	544.578	52
81	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	1.627.625	90
82	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	787.578	58
83	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	344.391	28
84	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	412.641	26
85	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	381.812	23
86	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	460.968	25
87	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	523.250	37
88	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	344.531	20
89	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	140.032	9
90	kth, nrt, roz, ppq, vzn, wzo, zeb,	7	591.797	26
91	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	862.672	52
92	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	786.453	66
93	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	461.938	42
94	kth, nrt, roz, ppq, vzn, wzo, zeb,	7	631.781	32
95	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	476.156	27
96	kth, nrt, roz, ppq, vzn, wzo, zeb,	7	387.828	12
97	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	522.312	39
98	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	543.766	43
99	chm, ngs, msv, pne, qry, roo, sbl, dvb, heg, knc, zne	11	947.188	82
100	air, bfo, bov, glk, kth, nrt, roz, ppq, vzn, zeb	10	397.078	30

Table C.1: Results of running the adapted Algorithm 2 on Problem Set 1.

C.2 Results for Problem Set 2

Run	Set	Set size	Time ms	Conversion steps
1	upf,uhu	2	56.406	12
2	xga,xma	2	54.859	12
3	upf,uhu	2	60.922	14
4	xga,xma	2	54.718	12
5	upf,uhu	2	54.422	12
6	upf,uhu	2	54.062	12
7	xga,xma	2	62.547	14
8	xga,xma	2	59.328	13
9	xga,xma	2	60.172	13
10	upf,uhu	2	58.032	13
11	xga,xma	2	62.922	14
12	xga,xma	2	62.625	14
13	upf,uhu	2	54.172	12
14	upf,uhu	2	54.328	12
15	upf,uhu	2	54.328	12
16	xga,xma	2	55.234	12
17	upf,uhu	2	54.391	12
18	xga,xma	2	59.156	13
19	upf,uhu	2	54.469	12
20	upf,uhu	2	58.219	13
21	upf,uhu	2	61.937	14
22	upf,uhu	2	61.250	14
23	xga,xma	2	59.687	13
24	xga,xma	2	61.282	14
25	upf,uhu	2	54.532	12
26	xga,xma	2	65.953	14
27	xga,xma	2	54.657	12
28	upf,uhu	2	58.843	13
29	upf,uhu	2	54.563	12
30	xga,xma	2	61.313	14
31	upf,uhu	2	57.546	12
32	xga,xma	2	54.531	12
33	xga,xma	2	57.437	12
34	upf,uhu	2	54.359	12
35	xga,xma	2	59.266	13
36	xga,xma	2	54.985	12
37	xga,xma	2	54.718	12
38	upf,uhu	2	58.375	13
39	xga,xma	2	54.000	12
40	xga,xma	2	55.172	12
41	xga,xma	2	82.938	19
42	upf,uhu	2	54.516	12
43	upf,uhu	2	61.625	14
44	xga,xma	2	58.968	13
45	xga,xma	2	62.734	14
46	upf,uhu	2	57.546	12
47	xga,xma	2	55.031	12
48	upf,uhu	2	56.984	12
49	xga,xma	2	54.266	12
50	xga,xma	2	74.438	15
51	upf,uhu	2	54.235	12
52	upf,uhu	2	54.406	12
53	xga,xma	2	60.140	13
54	upf,uhu	2	56.937	12
55	upf,uhu	2	57.234	12
56	xga,xma	2	55.187	12
57	xga,xma	2	58.469	12
58	xga,xma	2	57.594	12
59	upf,uhu	2	54.328	12
60	upf,uhu	2	64.937	14
61	xga,xma	2	58.375	12
62	xga,xma	2	54.782	12
63	upf,uhu	2	54.282	12
64	upf,uhu	2	57.922	13
65	upf,uhu	2	86.219	19
66	xga,xma	2	57.750	12
67	xga,xma	2	61.297	14
68	xga,xma	2	54.953	12
69	xga,xma	2	63.031	14
70	xga,xma	2	90.562	19

71	xga,xma	2	57.391	12
72	xga,xma	2	54.563	12
73	upf,uhu	2	54.188	12
74	xga,xma	2	63.046	14
75	xga,xma	2	54.953	12
76	xga,xma	2	58.266	13
77	upf,uhu	2	54.328	12
78	upf,uhu	2	61.688	14
79	upf,uhu	2	54.125	12
80	upf,uhu	2	57.281	12
81	xga,xma	2	58.812	13
82	xga,xma	2	58.813	13
83	xga,xma	2	62.750	14
84	upf,uhu	2	65.562	15
85	upf,uhu	2	78.796	19
86	upf,uhu	2	54.453	12
87	xga,xma	2	54.266	12
88	xga,xma	2	58.328	13
89	xga,xma	2	57.328	12
90	upf,uhu	2	57.594	13
91	upf,uhu	2	90.656	19
92	upf,uhu	2	66.969	15
93	upf,uhu	2	65.672	15
94	xga,xma	2	74.250	15
95	upf,uhu	2	61.687	14
96	xga,xma	2	59.938	13
97	upf,uhu	2	68.375	14
98	xga,xma	2	70.797	15
99	xga,xma	2	66.719	14
100	xga,xma	2	62.375	13

Table C.2: Results of running the adapted Algorithm 2 on Problem Set 2.

Bibliography

[1] S. Even, A. Itai, and A. Shamir, *On the complexity of timetable and multicommodity flow problems.* SIAM J Comput 5 (1976) 691–703.

[2] M.R. Garey, and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco, CA, USA: Freeman; 1979.

[3] R. Raghavjee, and N. Pillay, *A study of Genetic Algorithms to solve the school timetabling problem*, MICAI 2013, 2013 Proceedings Part II, pp. 64–80.

[4] G.H.G. Fonseca, H.G. Santos, and E.G. Carrano, *Late acceptance hill-climbing for high school timetabling*, Journal of Scheduling, 19 (2016) 453–465.

[5] E. Demirović, and N Musliu, *MaxSAT-based large neighborhood search for high school timetabling*, Computers Operations Research 78(2016) 172–180.

[6] S.S. Brito, G.H.G. Fonseca, T.A.M. Toffolo, H.G. Santos, and M.J.F. Souza, *A SA-VNS approach for the high school timetabling problem*, Electronic Notes in Discrete Mathematics 39 (2012) 169–176.

[7] Z. Lú, and J.K. Hao, *Adaptive tabu search for course timetabling*, European Journal of Operational Research 200 (2010) 235–244.

[8] N. Eén, and N. Sörensen, *An extensible SAT-solver*, Chalmers University of Technology, Sweden, 2003. `http://minisat.se/downloads/MiniSat.pdf`

[9] M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem-proving* Communications of the ACM, 5 (1962), 394–397.

[10] A. Biere, *Lingeling, Plingeling and Treengeling entering the SAT competition 2013* Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions, volume B-2013-1 of Department of Computer Science Series of Publications B, University of Helsinki (2013) pp. 51–52

[11] J. Marques Silva, and K. Sakallah, *GRASP—A new search algorithm for satisfiability* ICCAD '96: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, 1997, pp. 220–227.

[12] B.A.F.J. van Kesteren *The clustering problem in Dutch high schools: Changing metrics in search space*, Master's thesis, Leiden University, The Netherlands (1999).

[13] The GCC Team *GCC online documentation* `https://gcc.gnu.org/onlinedocs/`

[14] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, *Compiling finite linear CSP into SAT*, Constraints, 2 (2009) 254–272.

[15] N. Tamura, *Syntax of Sugar CSP description*, `http://bach.istc.kobe-u.ac.jp/sugar/package/current/docs/syntax.html`

[16] A. Biere, M. Heule, H. van Maaren, and T. Walsh (Eds.), Handbook of satisfiability, vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press; 2009.

[17] A. Schaerf, *A survey of automated timetabling*, CWI report CS-R9567, CWI, The Netherlands (1995).

[18] R.J. Willemen, *School timetable construction; Algorithms and complexity*, PhD Thesis, Technical University Eindhoven, The Netherlands (2002).

[19] N. Pillay, *A survey of school timetabling research*, Annals of Operations Research 218.1 (2014) 261–293.