Individual Study Activity (IADM801)

Solving the IMADA Timetabling Problem using Constraint-Based Local Search

Computer Science

Brian Alberg
brped13@student.sdu.dk

Supervised by Marco Chiarandini

April 23, 2019

SDU
UNIVERSITY OF
SOUTHERN DENMARK

Contents

1 Literature Review

Timetabling is an actively researched topic, and a problem which has to be resolved in many areas ranging from educational institutions to industry. Bettinelli et al. 2015 gave an overview of *curriculum-based course timetabling* (CB-CTT), and defined the problem in mathematical terms, along with possible variations and extensions to the problem. Bettinelli et al. 2015 also argued that the problem of timetabling is NP-hard, based on the problem's core, which resembles the *graph coloring* problem. Different techniques, *local search* being one of them, was presented and benchmarked on a number of different timetabling instances, but ultimately no "best" method was determined. It was argued that for CB-CTT in a university context, the problem differs slightly depending on the type of university. As an example, some universities might be divided into multiple departments at different locations, in which case the travel distance between locations have to be taken into consideration, while at single-building universities, this is less of a problem. It was argued that further research is needed on the topic of CB-CTT, and especially on which techniques are better, depending on the type of the timetabling problem.

A project where *local search* techniques showed promising results was in Gardi et al. 2014. Here the LocalSolver project, a solver for large-scale 0-1 non-linear programming, with a built-in modelling language was presented. While MILP solvers are used actively in industries today, because of their ease to use and model-and-run approach, these kinds of solvers will often give a *no solution found* on problems with a lot of variables, or spend a long time looking for an optimal solution. They found that industries are often just looking for a *good*, feasible solution in a short amount of time, and that solvers based on local search fit this profile more accurately. Furthermore LocalSolver showed good results at the ROADEF 2005 Challenge, addressing a real-life car sequencing problem proposed by the automotive company Renault. While brute force use of MILP solvers was unable to solve large instances of this problem, Gardi et al. 2014 developed a heuristic local search algorithm, which won, and ultimately ended up being deployed in Renault's 17 plants world-wide. Because of this there is reason to believe that *local search* could be perform well on many real-life problems, including those of in the class of CB-CTT.

Koch et al. 2011 presented a set of 361 instances, published with many details about each problem, and divided into groups according to their type and difficulty. Koch et al. 2011 states that commercial solvers

have become approximately 100 million times faster in the last 20 years, and since the release of the MIPLIB 2010 Problem Set, the academic community, industry, and every one else, have had the opportunity to test solvers in a standard way which is publicly available. Koch et al. 2011 also showed that the input format, e.g. the order of the constraints, can change the solution process dramatically, and cause unexpected changes in performance.

Because of these works we do not only have a class of problems and a general-purpose solver based on *constraint-based local-search*, but also a standardized way to benchmark and compare results.

2 Introduction

Timetabling is a task that has to be resolved at any educational institution, and even in many other workplaces such as those in the health care sector. It is however often a complicated task, especially if the resulting timetable has to be both feasible and balanced. The problem of timetabling can be formulated as a combinatorial optimization problem. When modelled as a mixed integer linear programming problem (MILP) the problem can be solved to optimality with state-of-the-art solvers. Nevertheless if the size of the problem is large enough, as it is the case for the scheduling of the courses at the faculty level, then the optimal solution becomes computationally hard. In this work, we try to solve the MILP model heuristically with local search.

In the context of a university, a timetable is a schedule divided into *weeks*, which are divided into *days*. These *days* are divided into sections of 1 hour each, which will be referred to as a *time slot*. A university has *courses*, which are taught at certain intervals, and has a number of students attending. In this report, an occurrence of a *course* in a specific *week* will be referred to as a *class*. Each *event* will be taught in a *room* by either a *teacher* or an *instructor*, depending on the type of event, which are *lectures* and *exercise sessions* respectively. An example of a 2-week schedule for a course with multiple events each week is shown in Table 1.

As seen in Table 1, some types of events take precedence to other types of events in the same course, meaning that they should be scheduled after a preceding event. For the course *C1*, introduction events should be scheduled before exercise events, and exercise events should be scheduled before laboratory events. Furthermore multiple sections, *H1* and *H2* are defined which correspond to multiple different exercise events. From this schedule, a *precedence graph* $G = (V, A)$ can be defined, as seen in

Course	Week	Type	Section	ID	
C1	37	Intro	F	C1-37-I-F1	← *event*
		Exercise	H1	C1-37-E-H1	
			H2	C1-37-E-H2	
		Lab	H1	C1-37-L-H1	
			H2	C1-37-L-H1	
	38	Intro	F	C1-38-I-F1	
				C1-38-I-F2	
		Exercise	H1	C1-38-E-H1	
			H2	C1-38-E-H2	

Table 1: Example of two weeks of *events* for a *course C1* with multiple classes of different type scheduled every week.

Figure 1.

Local Search is a heuristic method for solving computationally hard optimization problems. Local search algorithms move from solution to solution in a space of candidate solutions (called the search space) by applying local changes. The idea behind the optimization technique of *Constraint-based Local Search* (CBLS) is to apply these *local search* algorithms to the paradigm of *constraint programming*, wherein relations between variables are stated in the form of constraints. Compared to other optimization techniques CBLS algorithms sacrifice quality guarantees for performance, thus some times failing in finding optimal solutions. In some cases however, they might find high-quality or optimal solutions, even with short time limits. This makes CBLS ideal for large problems.

LocalSolver[1] is a general purpose local search solver, implementing a variety of techniques for solving optimization problems heuristically. According to the website of LocalSolver, "*it hybridizes different optimization techniques dynamically, during the resolution, thanks to a unique search approach*".

This report will present a definition of a specific CB-CTT problem, namely the IMADA Timetabling Problem, and how it was modelled and solved using LocalSolver, more specifically the LocalSolver C++ API. While LocalSolver is proprietary software, meaning that implementational details are limited, this report will mostly focus on the formulation of the IMADA Timetabling problem and how it is modelled in LocalSolver by the *Implementation*.

Finally the results will be compared to the results from a number of other

[1] https://www.localsolver.com

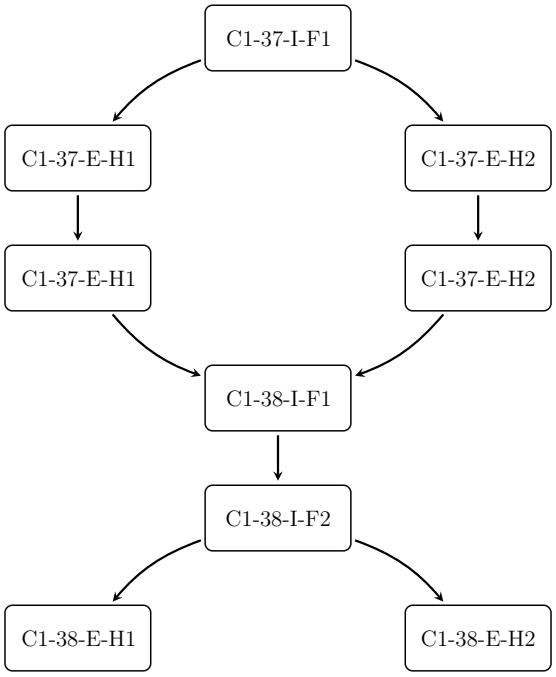

Figure 1: A precedence graph showing two weeks of *events* for course *C1*, and which events precede other events.

solvers, and these results will be discussed.

3 The IMADA Timetabling Problem

The IMADA Timetabling Problem is a problem where a number of events has to be scheduled into a number of time slots and suitable rooms. There is a number of *hard* constraints that must be satisfied for a feasible solution to exist, and a number of *soft* constraints which should be met but do not prevent a feasible solution.

The *hard* constraints are defined as follows:

H1 All events must be scheduled exactly once,

H2 Multiple courses may not occupy the same room at the same time slot,

H3 A room may not be assigned to a course if it is declared unavailable or if the room is unsuitable for the course,

H4 Events should not be scheduled in a time slot, where they would not have time to finish,

H5 Events should not be scheduled into time slots which are not allowed by calendar,

H6 A teacher may only teach one class at a time, and only if the teacher is available,

H7 For any course, only one event associated with that course must occur per day.

H8 For any course in each week *introduction* events precede *exercise* events, and *exercise* events precede *laboratory* events.

The *soft* constraints are defined as follows:

S1 Events of the same type should preferably be scheduled to the same day of the week and the same time slot of the day,

S2 Events of the same type should preferably be scheduled in the same room every week,

S3 A teacher should not teach more than one event per day,

S4 Each student should preferably not have more than 3 classes per day.

S5 Minimize the number of events occurring outside a normal work schedule (i.e. 9:00 to 17:00)

S6 Minimize the number of student overlaps, meaning the number of events a student has at any given point in time.

3.1 Constraints

The MILP formulated versions of the hard constraints are defined by a number of parameters. The parameters can be defined as:

E_w is a set of events in week w indexed by e,

C is a set of courses indexed by c,

R is the set of rooms indexed by r,

W is a set of weeks indexed by w,

$D = \{1 \dots 7\}$ is the set of days indexed by d,

H is the set of time slots on a given day and week. $H = \{8..17\}$

S is a set of students indexed by s,

T is a set of teachers indexed by t.

Furthermore, $x_{erwdh} \in \{0, 1\}$ defines if the given event $e \in E$ in room r starts in week w, day d and time slot h or not.

Constraints *H1* enforce that all events must be scheduled by stating that each event x_{erwdh} must start exactly once:

$$\sum_{d \in D} \sum_{h \in H} \sum_{r \in R} x_{erwdh} = 1 \qquad \forall e \in E_w,\, w \in W \tag{1}$$

Constraints *H2* enforce that multiple courses may not occupy the same room at the same time slot. If $a_{rwds} \in \{0, 1\}$ defines if a room r is available or not on week w, day d and time slot h, and $\ell(e)$ is defined as the duration (number of time slots) of an event e, then the constraints can be defined as:

$$\sum_{e \in E} \sum_{s=\max\{8, h-\ell(e)\}} x_{erwds} \leq a_{rwds} \qquad \forall r \in R,\, h \in H \tag{2}$$

Thus, for every room and time slot, the number of scheduled events in a room at a specific time slot, should be 0 if the room is unavailable or at most 1 if the room is available at the specific time slot. Hence, these constraints also enforce *H3*.

Since $h = \max\{8, s - \ell(e)\}$ and $s = 8, \ldots, 17$, these constraints also enforce *H4* and *H5*, since events cannot be scheduled e.g. at night, nor be scheduled to a time slot where they would not have time to finish.

Constraints *H6* enforce that a teacher may only teach one class at a time. If U_t is defined as a number of specific unavailabilities (w, d, h), meaning weeks, days and time slots where teacher t is unavailable, and E_t is the set of events for teacher t, the constraints *H6* can be enforced by not letting events taught by that teacher, scheduled into the time slots in U_t.

$$\sum_{e \in E_t} \sum_{r \in R} x_{erwdh} \leq 0 \qquad \forall t \in T,\, (w, d, h) \in U_t \tag{3}$$

However, this only restrict events in being scheduled into that time slot if the teacher is unavailable, but does not add any restriction in case the teacher is available. Thus, the following restricts the number of scheduled events at a specific time slot to one, if the teacher t is available:

$$\sum_{e \in E_t} \sum_{r \in R} x_{erwdh} \leq 1 \qquad \forall t \in T,\, (w, d, h) \notin U_t \tag{4}$$

Constraints *H7* enforce that for any course, only one event associated with that course occur per day. Thus, if E_c is the set of events for a specific course c, then for every event for that course, every day and every week, the sum of events for that course may not exceed 1. The constraints are defined as follows:

$$\sum_{e \in E_c} \sum_{r \in R} \sum_{h \in H} x_{erwdh} \leq 1 \qquad \forall e \in E_c,\, d \in D,\, w \in W,\, c \in C \qquad (5)$$

Constraints *H8* enforce the precedences of the different types of courses, such that for each week, some events of a course precede other events of the course. Let A be a set of ordered pairs of all events, such that each pair represents a preceding event and a succeeding event for a course. Then let f be a bijective function such that

$$f : W \times D \times H \to \mathbb{Z}^+$$

and

$$f^{-1} : \mathbb{Z}^+ \to W \times D \times H$$

and let P_w be a set of *periods* for week w such that week w weights the same as 5 days of 8 hours, and one day weights the same as 8 hours,

$$P_w = \{5 \cdot 8(w-1) + 8(d-1) + (h-1) \mid d \in D,\, h \in H\}.$$

The constraints *H8* can then be formulated as such:

$$\sum_{p \in P_w} p \sum_{r \in R} x_{irf^{-1}(p)} \leq \sum_{p \in P_w} p \sum_{r \in R} x_{jrf^{-1}(p)} \qquad \forall ij \in A,\, w \in W$$

which results in any event i to be scheduled before event j.

This can however be rewritten to

$$\sum_{d=1}^{7-d'} \sum_{h=8}^{17-\ell(j)} x_{irwdh} \leq x_{jrwd'h} \qquad \forall\, r \in R,\, ij \in A,\, w \in W \qquad (6)$$

which is a tighter version of the above since the summation is restricted to days instead of hours. Furthermore, the summation over the rooms R is moved out of the innter equation meaning that instead of checking

every room for every set of periods P_w, every set of periods is checked for every room.

3.2 Objectives

The objective of the MILP problem is constructed from the soft constraints. These are the constraints where violation should be minimized but should not prevent the solver from reaching a feasible solution.

For the constraints *S1*, the value of the objective function should be penalized if a course in one week is scheduled into a different day or time slot in other weeks. To do this, the number of discrepancies from a reference week is counted. If $\bar{w}(c)$ is defined as the reference week chosen at random among the weeks with the largest number of events requested for course c, E_{cw} is the set of events of course c in week w and $\bar{e}$ is the related event such that $\bar{e} \in E_{c\bar{w}_c}$, then the weekly stability can be defined as:

$$\sum_{e \in E_c} \sum_{w \in W} \sum_{(d,h) \in D \times H} \left| \sum_{r \in R} x_{erwdh} - \sum_{r \in R} x_{\bar{e}r\bar{w}dh} \right| \tag{7}$$

Thus, for each event in a course, each week, and each time slot in each day, the difference in time slot between the scheduled event and the reference event should be 0 if all events are scheduled in the same day and time slot every week, while differences will result in a higher value. This means that if (7) is minimized, the solver will try to enforce this constraint, and if this is not possible, the objective function value will be penalized for each discrepancy.

The same technique is used to guide the solver to use the same room every week for related events, as S2 enforces. This is defined as for each event in a course, each week and each room, the difference between the sum of scheduled events to that room and the sum of scheduled related events in the reference week.

$$\sum_{e \in E_c} \sum_{w \in W} \sum_{r \in R} \left| \sum_{(d,h) \in D \times H} x_{erwdh} - \sum_{(d,h) \in D \times H} x_{\bar{e}r\bar{w}dh} \right| \tag{8}$$

This difference will be 0 if both events are scheduled into the same room and > 0 otherwise, thus penalizing the solver for not assigning related events to the same room.

The constraints S3 tries to minimize that number of events per day per teacher. By defining E_t as the set of events for teacher t and assigning ϵ_{td} as the number of events per day d for teacher t, then for each day in each week, the number of scheduled events for that teacher on that day should be less than ϵ_{td}.

$$\sum_{e \in E_t} \sum_{r \in R} \sum_{h \in H} x_{erwdh} - 1 \leq \epsilon_{td} \qquad \forall t \in T, d \in D, w \in W$$

Thus, by minimizing

$$\sum_{t \in T} \sum_{d \in D} \epsilon_{td} \tag{9}$$

the number of events per day for any teacher will be minimized as well.

The constraints *S4* tries to guide the solver to avoid scheduling more than three classes per day per student. This is done by, for every student s and every day $d \in D$, δ_{sd} is assigned to be the number of events for student s on day d. The sum of all event on day d and week w for student s should be less than or equal to three. The sum of these events minus three, gives 0 if there are three events scheduled for that day, < 0 if less than three events are scheduled on that day, and > 0 if more than three events are scheduled.

$$\sum_{e \in E} \sum_{r \in R} \sum_{h \in H} x_{erwdh} - 3 \leq \delta_{sd}$$

By limiting to $\leq \delta_{sd}$, the number of courses per day per students should preferably be less than 3. By adding a weight, $\frac{1}{10}$, this solver is guided to deprioritize this constraint by one tenth compared to the other constraints.

$$\frac{1}{10} \left(\sum_{e \in E} \sum_{r \in R} \sum_{h \in H} x_{erwdh} - 3 \right) \leq \delta_{sd} \qquad \forall s \in S, d \in D, w \in W$$

Thus, by minimizing

$$\sum_{s \in S} \sum_{d \in D} \sum_{w \in W} \delta_{sdw} \tag{10}$$

the number of scheduled events per day per students should be minimized.

For every day a number of time slots can be defined as being *undesirable*, meaning that they are outside a normal work schedule, but that events should not be precluded from being scheduled into those slots. As an example, it should be possible to schedule classes between 16:00 and 18:00, but the solver should try to minimize the scheduling of events into these time slots, and try to schedule as many events into the time slots from 9:00 to 16:00 as possible.

Considering the constraints *S5*, by adding a weight θ to an event depending on the time slot the event is scheduled into, the solver can be guided to prefer scheduling into some time slots over others. Hence, let θ_p be a personalized weight depending on the slot, where

$$p \in P = \{(w, d, h) \mid w \in W,\, d \in D,\, h \in H\}$$

then the objective which should be minimized can be defined as:

$$\sum_{e \in E} \sum_{r \in R} \sum_{p \in P} \theta_p x_{erp} \tag{11}$$

The constraints *S6* adds the minimization of overlapping events for students, meaning the cases for any given point in time where a student have multiple events at the same time. If E_s is the set of events for student s, then the constraint can be defined as

$$\sum_{e \in E_s} \sum_{r \in R} x_{erwdh} - \gamma_{swdh} \leq 1 \qquad \forall w \in W,\, d \in D,\, h \in H,\, s \in S$$

Thus, by minimizing

$$\sum_{s \in S} \gamma_{swdh} \qquad \forall w \in W,\, d \in D,\, h \in H \tag{12}$$

the cases where a student have multiple courses at the same time will be minimized.

The problem (P) can be defined as a optimization problem with respect to all the above constraints. In this report Z will be refered to as the objective function of (P), $\vec{c} \in \mathbb{Z}$ is a vector of coefficients and $\vec{x} \in \mathbb{Z}$ is a vector of variables.

$$Z = \min \{(7) + (8) + (9) + (10) + (11) + (12) \mid (1), (2), \ldots, (6)\} \quad \text{(P)}$$

4 Relaxations

Since local search is not well-suited for hardly-constrained problems like the IMADA Timetabling Problem, relaxation techniques are implemented and tested. Each implementation is described in details below, and the results are discussed in the *Results and Discussion* section. From a user stand point, the relaxation technique is simply a parameter which can be set depending on the wanted relaxation, with one additional parameter, depending on the chosen relaxation technique.

The first technique described is *Lagrangian Relaxation*. Afterwards a *Lexicographic approach* is described, in which the solver will minimize the violations of a fraction of the constraints before solving the main objective.

4.1 Lagrangian Relaxation

Lagrangian Relaxation is a relaxation method in which a constraint can be moved to the objective function of a given problem such that the solver will not reject an infeasible solution in case the constraint cannot be satisfied. More specifically the idea is to remove one or more hard constraints from the problem at hand, and instead infer relaxations of the constraints to the objective function. When solving a problem which is relaxed using Lagrangian Relaxation, the solver will not reject an infeasible solution on the relaxed constraints, but will try to minimize (or maximize) the violations on the relaxed constraints.

One implemented technique using *Lagrangian Relaxation* is the relaxation of *H1*. This is added by omitting the addition of *H1* as constraints to the LocalSolver model, and instead adding the relaxation of it to the objective function via a vector of multipliers $\vec{\lambda}$. Thus we define a new problem (PR1):

$$
\begin{aligned}
Z_R = \min \quad & \left\{ \vec{c}^T\vec{x} + \vec{\lambda}\left|A\vec{x}' - \vec{b}'\right| \;\middle|\; \mathrm{S1}, \mathrm{S2}, \ldots, \mathrm{S4}, \mathrm{H2}, \mathrm{H3}, \ldots, \mathrm{H7} \right\} \\
= \min \quad & \vec{c}^T\vec{x} + \vec{\lambda}\left|\sum_{d\in D}\sum_{s\in S}\sum_{r\in R} x'_{erwdh} - 1\right| \quad \forall w \in W,\, e \in E \\
\text{s.t.} \quad & \mathrm{S1}, \mathrm{S2}, \ldots, \mathrm{S4} \\
& \mathrm{H2}, \mathrm{H3}, \ldots, \mathrm{H7}
\end{aligned}
\tag{PR1}
$$

The multipliers, or weights, $\vec{\lambda}$ are added such that for a *move*, meaning that with a change of one or more variables, the objective function will penalize

if the solver chooses to deprioritize the relaxed constraints $\left|A\vec{x} - \vec{b}\right|$, and prioritize the main objectives, while the objective function will reward the solver for prioritizing the relaxed constraints. In the optimal case the relaxed constraint will be equal to zero

$$\vec{\lambda}\left|A\vec{x} - \vec{b}\right| = 0$$

meaning that the relaxed constraint is solved. The vector $\vec{\lambda}$ is generated as follows.

Let μ be the sum of the four largest coefficients in the objective function Z_R, found by sorting the vector $\vec{c}^T$ in a non-increasing order, indicating the largest possible impact on the objective value on a move that changes four variables.

Let ϵ_i be the four largest coefficients in constraint a_i, where i is the index of the relaxed constraint, minus the bound b_i, indicating the largest impact on the constraint on a move that changes four variables.

$$\epsilon_i = \sum_{l=1}^{4} |a_i|^{(l)} \qquad \forall i \in M$$

In this case λ_i should be indicating that an improvement of the relaxed constraint is more important than an improvement on the main objectives, thus

$$|\mu| < \lambda_i \, |\epsilon_i| \qquad \forall i \in m$$

This indicates that λ_i is bounded by

$$\lambda_i > \frac{|\mu|}{|\epsilon_i|}$$

and that a suitable λ_i can be defined as

$$\lambda_i = \frac{|\mu|}{|\epsilon_i|} + 1$$

Since (PR1) contains less constraints than (P), LocalSolver will more easily be able to find a feasible solution for (PR1). Considering the relaxed constraints $\vec{\lambda}|A\vec{x} - \vec{b}|$, which was added to the objective function, the value

of the objective function will be penalized if $\vec{\lambda}|A\vec{x} - \vec{b}| > 0$, meaning that the relaxed constraint was not satisfied, thus letting $Z_R > Z$. However, if it was satisfied, meaning $\vec{\lambda}\left|A\vec{x}' - \vec{b}'\right| = 0$, the objective function will be such that $Z_R = Z$.

Relaxation of other types of constraints are defined as follows: For equality constraints of the type $A\vec{x} = b$, the relaxation R_e is defined

$$R_e = \vec{\lambda}\left|A\vec{x} - \vec{b}\right|$$

For inequality constraints of the type $A\vec{x} \leq b$, the relaxation R_l is defined

$$R_l = \vec{\lambda}\max(A\vec{x} - \vec{b}, 0)$$

For inequality constraints of the type $A\vec{x} \geq b$, the relaxation R_g is defined

$$R_g = \vec{\lambda}\max(\vec{b} - A\vec{x}, 0)$$

A relaxation technique using Lagrangian Relaxation is implemented. Here Lagrangian Relaxation is performed on a fraction φ of the constraints, where $0 \leq \varphi \leq 1$. Given a problem of m constraints, the number of constraints to be relaxed is

$$k = m\varphi.$$

These relaxations are added via the multipliers $\vec{\lambda}$ for the k constraints of given instance.

Using this technique, the first k relaxations will be added to the objective function. Afterwards the remaining $m - k$ constraints will be added to the model as regular constraints.

The resulting problem can be defined as problem (PR2):

$$\begin{aligned} Z_R = \min \quad & \vec{c}^T\vec{x} + R_e + R_l + R_g \\ \text{s.t.} \quad & \textit{remaining } m - k \textit{ constraints} \end{aligned} \tag{PR2}$$

4.2 Lexicographic Approach of LocalSolver

In the *lexicographic approach*, relaxation is performed on a fraction φ of the constraints, where $0 \leq \varphi \leq 1$. Like in the *Lagrangian Relaxation*, the number of constraints to be relaxed is

$$k = m\varphi.$$

Likewise, relaxations are defined with respect to the type of constraint, but in contrast to *Lagrangian Relaxation* the weight $\vec{\lambda}$ is not added. Thus, for the first k constraints, the following relaxations are defined:

1. $R_e = \left|A\vec{x} - \vec{b}\right|$ for constraints of form $A\vec{x} = \vec{b}$,
2. $R_l = \max(A\vec{x} - \vec{b}, 0)$ for constraints of form $A\vec{x} \leq \vec{b}$,
3. $R_g = \max(\vec{b} - A\vec{x}, 0)$ for constraints of form $A\vec{x} \geq \vec{b}$,

A new objective function G is defined, that contain the relaxations of the first k constraints:

$$G = R_e + R_l + R_g.$$

Thus a new problem (PR3) can be defined as:

$$\begin{aligned} Z_R = \min \quad & (G(\vec{x}), Z(\vec{x})) \\ \text{s.t.} \quad & \textit{remaining } m - k \textit{ constraints} \end{aligned} \tag{PR3}$$

As this new problem (PR3) contains two objective functions, the multi-objective functionality of LocalSolver is utilized. LocalSolver supports multiple objective functions, by exposing methods for adding these to the model. When solving starts, LocalSolver will evaluate all the objectives lexicographically in each iteration, and prioritize moves that improve on the early objective functions (meaning those that was added to the model first).

Thus, by adding G to the model before Z, in the problem (PR3), improvements on G will take priority over improvements on Z. Ultimately this means that LocalSolver will attempt to satisfy the relaxed constraints over solving the main objective function, which is the desired effect.

5 Implementation

An implementation was made in C++, which reads an instance of the IMADA Timetabling Problem given as a file in MPS-format[2] (Mathematical Programming System). The MPS file was generated from an existing framework, which beside from solving the IMADA instance using SCIP,

[2] http://lpsolve.sourceforge.net/5.5/mps-format.htm

also handles reading of data and generation of a model as LP and MPS format.

CoinMpsIO from the Coin-OR[3] project was utilized to parse the MPS file and store the instance in memory. After reading the file, a model is generated using LocalSolver C++ API, and LocalSolver is told to solve the model. The result is returned as a plain-text file containing the result of the objective function and the result of all variables returned by LocalSolver.

The implementation can be narrowed down to the following steps:

1. read MPS file containing the instance,
2. declare the MILP instance to LocalSolver using the C++ API,
3. solve the instance using LocalSolver,
4. output the result to a file.

The central part of the implementation is the declaration of the MILP instance to LocalSolver using the C++ API. This part of the implementation is in itself a number of steps, as shown in Figure 2.

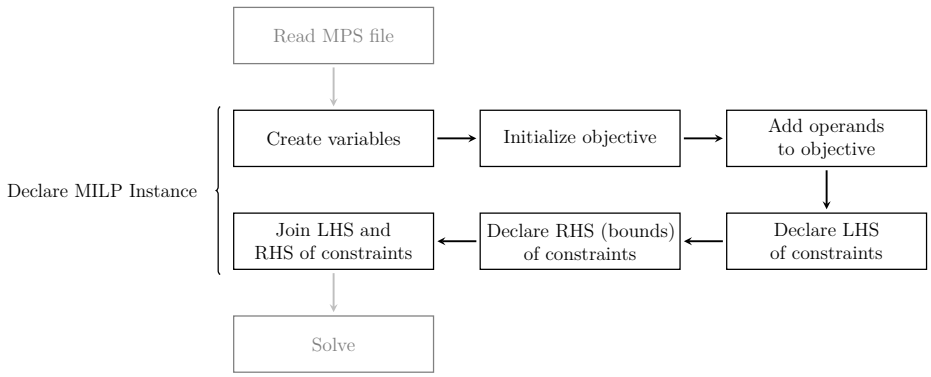

Figure 2: The steps of the implementation that declares the MILP instance to LocalSolver using the C++ API.

Since the last step, *Join LHS and RHS*, is the step that takes care of the relaxation, it is also the most complex of the steps, and will be described here in details. The method initializing this step is the `bindAllConstraints` function, which takes the relaxation method, `r_method` and the fraction of constraints to be relaxed `relax_frac` as parameters. The source code is shown in Listing 1, and starts by finding the number of constraints to be relaxed `num_relax`. Then, depending on the relaxation method,

[3] https://www.coin-or.org/

other methods is called. In case the relaxation method is using the Lagrangian Relaxation technique, one of the `findLambdas` methods is called, and then the corresponding `joinConsAndBounds` method, while if using the Lexicographic technique or no relaxation, only the corresponding `joinConsAndBounds` method is called.

```
void LSSolver::bindAllConstraints(RelaxMethod r_method, double relax_frac) {
    this->relax_method = r_method;
    vector<int> lambdas;

    // Find number of constraints to be relaxed
    int num_relax = round(constraints.size()*relax_frac);

    // Add cons and bounds
    if(this->relax_method == ESS1) {
        lambdas = findLambdasESS1();
        joinConsAndBoundsESS1(lambdas);
    } else if(this->relax_method == LEX) {
        joinConsAndBoundsLEX(num_relax);
    } else if(this->relax_method == LGR) {
        lambdas = findLambdasLGR(num_relax);
        joinConsAndBoundsLGR(lambdas);
    } else {
        joinConsAndBoundsPlain();
    }
}
```

Listing 1: Function for handling the relaxation and joining left-hand sides and right-hand sides of constraints together.

In the case where one of the `findLambdas` methods is called, using `findLambdasLGR` as an example, the function will take the number of constraints to be relaxed as a parameter `num_relax`. The function will then start by finding the highest change in the objective function by first making a copy of the objective coefficients, then sorting the copy in non-increasing order and then store the highest possible change in the objective function in the case where 3 variables are changed in one move to `max_obj`. This is shown in Listing 2.

Then for each constraint to be relaxed, the same is done for the RHS of the constraints. The function stores a vector containing the names of the constraints of which the lambda was calculated, which is used to ignore dublicate constraint declarations in the MPS file. Constraints which have already been considered are simply ignored. Next, the bound

```
vector<int> LSSolver::findLambdasLGR(unsigned int num_relax) {
    vector<int> lambda;
    lambda.reserve(num_relax);
    int max_con_change = 0;
    unsigned int flips_per_iter = 3;
    vector<string> lambda_names;

    // Find highest change in objective function
    vector<int> sorted_objcoeff = objcoeff.at(0);
    sort(sorted_objcoeff.rbegin(), sorted_objcoeff.rend());
    int max_obj = 0;
    for(unsigned int i = 0; i < flips_per_iter; i++) {
        max_obj += abs(sorted_objcoeff[i]);
    }
    ...
```

Listing 2: Part (1/2) of the function for calculating $\vec{\lambda}$ if using the Lagrangian Relaxation technique, showing how the highest possible change of the objective function is calculated.

of the constraint is considered. Every constraint has both an upper and lower bound, even if one of them is unbounded (in which case it is equal to the highest or lowest possible value respectively). Thus, the function will consider if the upper or lower bound should be used to calculate the lambda, and the lambda is calculated and saved to the `lambda` vector

```
lambda.push_back(abs((abs(max_obj)/abs(max_con_change))+1));
```

This corresponds to description in the *Relaxation* section, where $\lambda_i = \frac{|\mu|}{|\epsilon_i|} + 1$. This is shown in Listing 3. Finally the lambda vector is returned, corresponding to $\vec{\lambda}$.

Afterwards the corresponding `joinConsAndBounds` method is called. This method will add the relaxations of the constraints depending on the method, and add the remaining constraints as hard constraints. As the constraints are actually stored in two seperate vectors (`constraints` and `bounds`) containing the left-hand side and the right-hand side of the constraint, these have to be joined into actual constraints before they are added to the LocalSolver model. This is shown in Listing 4, 5 and 6.

Details on why the left-hand side and right-hand side are seperated, and thus why they have to be joined can be found in Appendix A.

```
    ...
    unsigned int i = 0;
    while(i < num_relax) {
        if(find(lambda_names.begin(), lambda_names.end(),
         ↪  bounds_v.at(i).name) == lambda_names.end()) {
            if(coefficients.at(i).size() >= flips_per_iter) {
                // Find highest change in constraint
                vector<int> con_coeff = coefficients.at(i);
                sort(con_coeff.rbegin(), con_coeff.rend());

                for(unsigned int j = 0; j < flips_per_iter; j++) {
                    max_con_change += abs(con_coeff[j]);
                }
            }

            // Check if we should use lower or upper bound
            int bound;
            if((char)bounds_v.at(i).type == 'L' ||
             ↪  (char)bounds_v.at(i).type == 'R') {
                bound = bounds_v.at(i).ub;
            } else {
                bound = bounds_v.at(i).lb;
            }
            max_con_change = abs(max_con_change - bound);
            if(max_con_change == 0) { max_con_change = 1; }
            lambda.push_back(abs((abs(max_obj)/abs(max_con_change))+1));
            max_con_change = 0;
            lambda_names.push_back(bounds_v.at(i).name);
            i++;
        }
    }
    return lambda;
}
```

Listing 3: Part (2/2) of the function for calculating $\vec{\lambda}$ if using the Lagrangian Relaxation technique, showing how the highest possible change of the objective function is calculated.

```
void LSSolver::joinConsAndBoundsLGR(vector<int> lambda) {
    unsigned int num_relax = lambda.size();
    LSExpression relax_obj = model.sum();
    relax_obj.setName("relax_obj");
    unsigned int k = 0;

    for(unsigned int i = 0; i < constraints.size(); i++) {
        unsigned int b = i;

        // Find correct bound by comparing names of lhs and rhs
        while(bounds_v.at(b).name.compare(constraints.at(i).getName()) !=
         ↪  0) {
            b++;
            if(b >= bounds_v.size()) { b = 0; }
        }

    if(k < num_relax) {
        // See Listing 5
    } else {
        // See Listing 6
    }
  }
  objectives.push_back(relax_obj);
  objectives.at(0) += relax_obj;
}
```

Listing 4: The `joinConsAndBounds` method for *Lagrangian Relaxation*. This method joins the left-hand side and right-hand side of a constraint together.

```
switch((char)bounds_v.at(i).type) {
    case 'E':
        relax_obj += lambda[k] * model.abs(constraints.at(i) -
         ↪  bounds_v.at(i).lb);
        break;
    case 'G':
        relax_obj += lambda[k] * model.max(bounds_v.at(i).lb -
         ↪  constraints.at(i), 0);
        break;
    case 'L':
        relax_obj += lambda[k] * model.max(constraints.at(i) -
         ↪  bounds_v.at(i).ub, 0);
        break;
    case 'R':
        relax_obj += lambda[k] * model.max(bounds_v.at(i).lb -
         ↪  constraints.at(i), 0);
        relax_obj += lambda[k] * model.max(constraints.at(i) -
         ↪  bounds_v.at(i).ub, 0);
        break;
}
k++;
```

Listing 5: If the number of constraints to be relaxed has not been reached yet, relax the constraint.

```
switch((char)bounds_v.at(i).type) {
    case 'E':
        model.constraint(constraints.at(i) == bounds_v.at(i).lb);
        break;
    case 'G':
        model.constraint(constraints.at(i) >= bounds_v.at(i).lb);
        break;
    case 'L':
        model.constraint(constraints.at(i) <= bounds_v.at(i).ub);
        break;
    case 'R':
        model.constraint(bounds_v.at(i).lb <= constraints.at(i)
         ↪  <= bounds_v.at(i).ub);
        break;
}
```

Listing 6: If the number of constraints to be relaxed has been reached, add the constraint as a hard constraint.

6 Results and Discussion

6.1 Test Results

The two general-purpose relaxation techniques, *Lagrangian Relaxation* (LGR) and *Lexicographic Approach of LocalSolver* (LEX), were tested for a fraction of the constraints on a number of *binary problems* from the MIPLIB 2010 Problem Set[4]. The fraction to be relaxed was set to $\varphi = 0.1$. A plain method using LocalSolver (without any relaxation) was tested as well, and tests on the same instances using SCIP was made for comparison. In all cases, a time limit was set to 120 seconds, and the number of threads to be used was set to 8. The full results are given in the Results Section. The listed results in Table 3, 4 and 5 are the average results for 3 runs on each instance, using the two different relaxation techniques, 3 runs on each instance using no relaxation technique and 3 runs on each instance using SCIP. The *Optimal* solution, number of *Rows* and number of *Columns* listed is from the MIPLIB 2010 Problem Set.

Furthermore, results were compared to results from a *General-Purpose Local Search solver* made as a thesis project by a previous student at The University of Southern Denmark, results from simple local search solvers made by students in the course DM841 *Heuristics and Constraint Programming for Discrete Optimization* at the University of Southern Denmark, along with a base-line algorithm that starts from a random state and applies a first improvement local search in the flip neighborhood, also on a subset of the MIPLIB 2010 Problem Set.

LSPlain Refers to the results from LocalSolver where no relaxation technique was used.

LSLGR Refers to LocalSolver with the *Lagrangian Relaxation of a Percentage of the Constraints* relaxation technique applied, and lists the *result* of the objective function and *violations* of the relaxed constraints.

LSLEX Refers to the *Lexicographic Relaxation of LocalSolver* relaxation technique, and lists the *result* of the main objective function Z and the sum of the *violations* of the relaxed constraints, corresponding to the result of objective function G.

SCIP Refers to the results of tests made with The SCIP Optimization Suite 6.0.0 (A. Gleixner et al. 2018). Because of the relatively short time limit of 120 seconds, SCIP would spend most of the time *pre-solving* on many of the test sets. Thus, the parameter.

[4] http://miplib.zib.de/miplib2010.php

`presolving/abortfac = 0.01`

is added to limit SCIP to only pre-solve as long as the minimum decrease of the problem size per round is greater than or equal to 1% of the problem size.

GLS Refers to the results of a *General-Purpose Local Search solver* made as a thesis project by a previous student at The University of Southern Denmark.

Best DM841 Refers to the results of the best local search solver from the course DM841 *Heuristics and Constraint Programming for Discrete Optimization* at the University of Southern Denmark.

LF+FI Refers to a base-line algorithm that starts from a random state and applies a first improvement local search in the flip neighborhood.

The IMADA Timetabling was attempted to be solved using LSPlain, LSLGR, LSLEX and the *Lagrangian Relaxation* technique only on the constraints *H1*. The same parameters were used for these tests.

The listed violations (*Vio.*), refers to the sum of the relaxed constraints, which for the techniques based on *Lagrangian Relaxation* is also included in the *Result*, since the relaxed constraints is added to the objective function.

The hardware used was a Asus ROG G56JK laptop with an Intel i7-4700HQ Quad-core CPU @ 2.40Ghz with Hyper-threading enabled and 12 GB DDR3 RAM.

6.2 Evaluation

The table below shows statistics for the results of each relaxation technique.

Optimal The number of instances where the objective value is equal to the optimal objective value and the number of violations is equal to zero.

Feasible The number of instances where the objective value is not equal to the optimal objective value, but the number of violations is equal to zero. Note that this is a superset of *Optimal*.

Relaxed Violated The number of instances where the sum of violations on the relaxed constraints is higher than zero, but the hard constraints were satisfied.

Hard Violated The number of instances where the constraints that was not relaxed (hard constraints) were violated.

	SCIP	LSPlain	LSLGR	LSLEX
Optimal	14	9	13	10
Feasible	46	41	38	38
Relaxed Violated	–	–	23	21
Hard Violated	26	31	11	13

Table 2: Statistics for the results of runs on the whole subset of the MIPLIB 2010 Problem set where results were present. The numbers indicate the number of results from instances the that fit the category.

From Table 2 it is seen that LSPlain was able to find a feasible solution for 57% of the test sets, with 12.5% of the total number of tests being optimal and 43% giving infeasible results.

Using the *Lagrangian Relaxation* technique, LSLGR was able to find a feasible solution for 53% of the test sets, with 18% of the total number of tests being optimal, and 47% giving infeasible results, where 69% of these were because of violations of the relaxed constraints.

Using the *Lexicographic Approach of LocalSolver*, LSLEX, 53% of the tests gave a feasible solution, with 14% of the total number of tests being optimal. 47% of the tests gave infeasible results where 29% of the total was due to violations of the relaxed constraints.

By comparison, SCIP was able to find an optimal solution for ~19% of the instances, with ~64% resulting in a feasible solution, and 36% being infeasible.

In Table 6 the results from the runs on the IMADA Timetabling Problem is shown. Only SCIP was able to find a feasible solution. LocalSolver using any of the relaxed techniques all gave results with violations on the relaxed constraints only, while LocalSolver without any relaxation did not reach any solution.

In Table 7 all of the LocalSolver methods and SCIP is compared to a local search solver from the DM841 course at the University of Southern Denmark. The solver was the best among 22 other local search solvers which were handed in as assignments for the course. Furthermore a local search baseline algorithm is shown (LS+FI), that starts from a random state and applies a first improvement local search on the flip neighborhood. Only as small subset of the MIPLIB 2010 problem set is compared to, because of restricted amount of data available for these methods. These

results show that LSPlain is only able to reach an optimal solution on 1 of the 9 instances, and a feasible solution on 2 of the 9 instances. This is the same for LocalSolver using *Lexicographic Approach of LocalSolver*. Using the relaxation technique LSLGR an optimal solution was reached on 2 of the instances, and a feasible solution on 1 additional instance. SCIP was able to reach an optimal solution on 3 of the instances, and a feasible solution on 1 additional instance. LS+FI and the best solver from the DM841 course was not able to find any optimal solutions, but found each 2 feasible solutions.

In Table 8 LocalSolver, SCIP and GLS is compared on a subset 38 instances of the MIPLIB 2010 Problem Set. Of these, GLS was able to find feasible solutions for 16 of the instances, which only 1 being the optimal solution. SCIP was able to find 22 feasible solutions, 13 of these being optimal. LSPlain was able to find 19 feasible solutions with 8 being optimal, LSLGR was able to find 19 feasible solutions with 11 being optimal and LSLEX was able to find 18 feasible solutions with 8 being optimal.

Thus, SCIP looks to be the clear winner, with the LocalSolver techniques being second. However, looking further into the number of violations of the constraints for each of the instances and solvers, both the GLS solver and the *best DM841* solver often violate the constraints by far less than any of the LocalSolver techniques. This may or may not indicate that the GLS Solver, the best solver from the DM841 course and LocalSolver is actually not far from each other in terms of performance.

Furthermore it can be concluded that relaxing constraints did not give considerably better results compared to the tests where no constraints was relaxed. While the tests using LSLGR resulted in slightly more *Optimal* solutions than any of the other approaches, LSPlain gave the highest number of feasible solutions. Furthermore, since the majority of the tests which gave an infeasible solution using the two relaxation techniques was due to violations on the relaxed constraints, it can be argued that the relaxation techniques did not fulfill their purpose in trying to guide the solver to minimize the sum of the relaxed constraints. For some of the problems it could be argued that the constraints to be relaxed is coincidentally the most difficult subset of the constraints. However, this would be a generalization that would be difficult to prove. Instead it could be argued that for the *Lagrangian Relaxation* technique, this could be because of a bad λ being calculated. For the *Lexicographic Approach* it is difficult to assume anything because of the proprietary nature of LocalSolver.

Meanwhile, SCIP *did* give considerably better results compared to any of the techniques using LocalSolver, with SCIP outperforming the *Lagrangian Relaxation* technique, which got the highest number of *optimal* solutions by 4 percentage points, and the *plain* LocalSolver in number of *feasible* solutions by 7 percentage points. While the techniques used by LocalSolver and SCIP is quite different which would directly affect this comparison in itself, another difference is noteworthy. SCIP is Open Source Software and developed since 2005 with a lot of contributions from both the academic community and industry, while LocalSolver is proprietary software and is developed since 2007. Thus LocalSolver have not only had 2 years less on the market than SCIP, but is presumably also developed by a considerably smaller team.

Another thing worth noting is that the performance of LocalSolver is actually not far from the performance of the best solver from the DM841 course, and the GLS solver. Both of these solvers was made by master students at the University of Southern Denmark, one of them being made as an assignment for a course and the other being made as a master's project.

7 Conclusion

While LocalSolver provides an easy-to-use API with a lot of features, the results from the implementation using LocalSolver's C++ API does not look promising. As LocalSolver is a solver which prides itself to be a *premier all-terrain & all-in-one optimization solver*, it is surprising that the solver was only able to solve 57% of the test cases, with only ~13% of the total number of test cases giving an optimal solution. While the number of optimal solutions found was slightly increased using the *Lagrangian Relaxation* technique, this decreased the number of feasible solutions found, thus it could be argued that the relaxation techniques did not improve the results enough to be conclusive. Instead it could be suggested that while LocalSolver might work perfectly fine for less difficult, real-life problems, it does not handle problems of high difficulty well, at least not out-of-the-box. However, as there does not seem to be any correlation between the number of variables and constraints, and the result found by LocalSolver, it is difficult to say exactly when LocalSolver finds a problem to be too difficult. It was also shown that LocalSolver was not able to outperform The SCIP Optimization Suite when it comes to *general-purpose* solving of problems, for which SCIP was the best choice.

Meanwhile, results from non-commercialized solvers, made as relatively small projects by students in the recent years, came quite close to the performance of LocalSolver in terms of minimizing the violations of the constraints. After looking at all the results from the DM841 course, which is not included in this report, it was could even be suggested that the combined effort of the 22 solvers gave better results than that of LocalSolver.

As for the IMADA Timetabling problem, it is disappointing that LocalSolver was not able to find an feasible solution using any of the approaches. However, this instance did come from another source, mainly from the framework for solving the IMADA Timetabling problem using SCIP, and as argued in Koch et al. 2011, the order in which the constraints is handed to the solver is important. Whether this influenced the results of the IMADA Timetabling problem or not is unknown, but there is a possibility that the instance contains types of constraints or other oddities which is not handled optimally by the implementation. Furthermore, the instance of the IMADA Timetabling problem is the only instance which contains variables which are integers, and thus the problem is not a *binary* problem like the instances used from the MIPLIB 2010 problem set.

While more analysis could have been done on the results in this report, it was simply not possible because of time constraints, and will be left open

for future research. However, the results also gives a clear impression that LocalSolver is not as good as claimed, considering the small gap in performance between non-advanced, student-made local search solvers. This opens up for future work in the area of Constraint-based Local Search solvers and their performance, especially when it comes to *which* classes of problems this type of solver is able to solve, and *when* a problem becomes too difficult.

It is also clear that further research is needed in the area of Constraint-based Local Search when used for solving CB-CTT problems.

References

Bettinelli, Andrea, Valentina Cacchiani, Roberto Roberti, and Paolo Toth (2015). "An overview of curriculum-based course timetabling". In: *Top* 23.2, pp. 313–349.

Gardi, Frédéric, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel (2014). *Mathematical Programming Solver Based on Local Search*. John Wiley & Sons.

Gleixner, Ambros, Michael Bastubbe, Leon Eifler, et al. (July 2018). *The SCIP Optimization Suite 6.0*. Technical Report. Optimization Online. URL: http://www.optimization-online.org/DB_HTML/2018/07/6692.html.

Koch, Thorsten, Tobias Achterberg, Erling Andersen, et al. (2011). "MIPLIB 2010". In: *Mathematical Programming Computation* 3.2, p. 103.

8 Results

Problem				SCIP	LSPlain	LSLGR		LSLEX	
Name	Rows	Columns	Optimal	Result	Result	Result	Viol.	Result	Viol.
air04	823	8904	56137	56137	–	56137	0	–	62
cov1075	637	120	20	20	20	20	0	20	0
ex1010-pi	1468	25200	–	253	240	239	0	241	0
go19	441	441	84	84	84	84	0	84	0
iis-100-0-cov	3831	100	29	29	29	29	0	29	0
iis-bupa-cov	4803	345	36	36	36	36	0	36	0
iis-pima-cov	7201	768	33	33	33	33	0	33	0
m100n500k4r1	100	500	-25	-24	-24	-24	0	-24	0
macrophage	3164	2260	374	374	409	421	0	419	0
methanosarcina	14604	7930	–	2893	2894	2887	0	2893	0
n3div36	4484	22120	130800	135000	131000	136400	0	134600	0
n3seq24	6044	119856	52200	52260400	69000	77600	0	70200	0
neos-1109824	28979	1520	–	378	404	558	84	668	0
neos-1337307	5687	2840	-202319	-202204	-201584	-1190450	1775	-205386	9
neos-1440225	330	1285	36	36	–	36	0	–	0
neos-1616732	1999	200	159	160	159	159	0	159	0
neos-1620770	9296	792	9	9	9	9	0	9	0
neos-631710	169576	167056	203	215	435	390	0	398	0
neos-777800	479	6400	-80	-80	-80	-80	0	-80	0
neos-941313	13189	167910	9361	–	21087	20459	2811	19111	0
neos18	11402	3312	16	16	17	17	0	17	0
neos808444	18329	19846	0	–	–	0	0	0	0
ns1853823	224526	213440	–	3560000	3019333	1572000	0	2158000	0
opm2-z10-s2	160633	6250	-33826	-18201	-25443	-13834	0	-22026	0
opm2-z11-s8	223082	8019	-43485	-27495	-32728	-13624	0	-22696	0
opm2-z12-s14	319508	10800	-64291	-34409	-47704	-19769	0	-25037	0
opm2-z12-s7	319508	10800	-65514	-37611	-47098	-20078	0	-20972	0
opm2-z7-s2	31798	2023	-10280	-9922	-8673	-8205	0	-8724	0
p6b	5852	462	-63	-62	-62	-63	0	-63	0
pb-simp-nonunif	1451912	23848	–	89	87	2305	2304	1	576
queens-30	960	900	-40	-33	-39	-39	0	-39	0
ramos3	2187	2187	–	242	224	220	0	224	0
seymour	4944	1372	423	427	424	424	0	424	0
seymour-disj-10	5108	1209	287	296	289	289	0	290	0
sp97ar	1761	14101	660705645.76	780606820	723269000	714205000	0	713828000	0
sp98ic	825	10894	449144758.4	531907348	473185000	474335000	0	464611000	0
sts405	27270	405	–	350	341	342	0	341	0
sts729	88452	729	–	657	648	647	0	646	0
t1717	551	73885	–	235124	–	326893	12166	339117	0
t1722	338	36630	–	165792	200739	177635	0	190983	1
tanglegram2	8980	4714	443	443	443	443	0	445	0
toll-like	4408	2883	610	621	658	671	0	657	0
vpphard	47280	51471	5	–	205	44	36	15	12
vpphard2	198450	199999	81	–	881	177	166	133	93
wnq-n100-mw99-14	656900	10000	259	–	2196	1893	0	2201	0

Table 3: Results from runs where at least one of the LocalSolver (LS) methods was able to reach a **Feasible** solution

Problem				SCIP	LSPlain	LSLGR		LSLEX	
Name	Rows	Columns	Optimal	Result	Result	Result	Viol.	Result	Viol.
acc-tight4	3285	1620	0	–	–	16	16	0	44
acc-tight5	3052	1339	0	0	–	42	42	0	49
acc-tight6	3047	1335	0	–	–	37	37	0	45
bley_xl1	175620	5831	190	–	–	68281	68056	230	40
circ10-3	42620	2700	–	–	–	599	367	311	20
co-100	2187	48417	2639942.06	12931802	–	985028000	983827579	4723870	52
mspp16	561657	29280	363	–	–	358	17	477	8
neos-506428	129925	42981	583780	–	–	570149	424204	0	95
ns1688347	4191	2685	27	27	–	105	104	1	2
ns1696083	11063	7982	45	–	–	301	274	10	4
ns894236	8218	9666	–	–	–	58	40	18	20
ns894244	12129	21856	15	–	–	172	156	16	56
ns894786	16794	27278	–	–	–	158	144	14	45
ns894788	2279	3463	7	–	–	15	8	7	3
ns903616	18052	21582	19	–	–	214	192	22	57
protfold	2112	1835	–	-12	–	-31	54	-16	5

Table 4: Results from runs where solutions from neither of the LS methods was **Feasible** because of violations of the relaxed constraints

Problem				SCIP	LSPlain	LSLGR		LSLEX	
Name	Rows	Columns	Optimal	Result	Result	Result	Viol.	Result	Viol.
bnatt350	4923	3150	0	–	–	–	45	–	32
bnatt400	5614	3600	1	–	–	–	48	–	32
datt256	11077	262144	–	–	–	–	149	–	94
ex10	69608	17680	100	–	–	–	60	–	14
ex9	40962	10404	81	–	–	–	76	–	16
f2000	10500	4000	–	–	–	–	306	–	163
hanoi5	16399	3862	1931	–	–	–	1460	–	447
neos-849702	1041	1737	0	–	–	–	–	–	–
netdiversion	119589	129180	242	4900438	–	–	245598363	–	1699
ns1663818	172017	124626	86	–	–	–	71455	–	440
tanglegram1	68342	34759	5182	5431	–	–	54	–	415

Table 5: Results from runs where solutions from neither of the LS methods was **Feasible** because of violations of the hard constraints

Problem				SCIP	LSPlain	LSLGR		LSLEX		LS Relax H1	
Name	Rows	Columns	Optimal	Result	Result	Result	Vio.	Result	Vio.	Result	Vio.
imada	133111	103547	485	485	–	8868	7480	96636800000	232	10118	8096

Table 6: Results from runs on the IMADA instance

Problem				Best DM841		LS+FI		SCIP	LSPlain	LSLGR		LSLEX	
Name	Rows	Columns	Optimal	Result	Vio.	Result	Vio.	Result	Result	Result	Vio.	Result	Vio.
acc-tight6	3047	1335	0	0	3	0	88	–	–	37	37	0	45
bnatt350	4923	3150	0	0	122	0	534	–	–	–	45	–	32
co-100	2187	48417	2639942.06	2220	82	194	1201	12931802	–	985028000	983827579	4723870	52
ex10	69608	17680	100	97	61	645	967	–	–	–	60	–	14
ex9	40962	10404	81	80	46	202	430	–	–	–	76	–	16
iis-pima-cov	7201	768	33	34	0	77	0	33	33	33	0	33	0
macrophage	3164	2260	374	436	0	800	0	374	409	421	0	419	0
neos-1440225	330	1285	36	36	37	46	107	36	–	36	0	–	0
ns1696083	11063	7982	45	43	3	54	1541	–	–	301	274	10	4

Table 7: Comparison with the best solver from the DM841 course and a baseline algorithm (LS+FI) that starts from a random state and applies a first improvement local search in the flip neighborhood on a subset of the MIPLIB 2010 set

Problem				GLS		SCIP	LSPlain	LSLGR		LSLEX	
Name	Rows	Columns	Optimal	Result	Vio.	Result	Result	Result	Vio.	Result	Vio.
acc-tight4	3285	1620	0	0	3	–	–	16	16	0	44
acc-tight5	3052	1339	0	0	10	0	–	42	42	0	49
acc-tight6	3047	1335	0	0	7	–	–	37	37	0	45
air04	823	8904	56137	71884	54	56137	–	56137	0	–	62
bnatt350	4923	3150	0	–	–	–	–	–	45	–	32
co-100	2187	48417	2639942.06	–	–	12931802	–	985028000	983827579	4723870	52
cov1075	637	120	20	21	0	20	20	20	0	20	0
ex10	69608	17680	100	98	66	–	–	–	60	–	14
ex9	40962	10404	81	81	42	–	–	–	76	–	16
go19	441	441	84	88	0	84	84	84	0	84	0
hanoi5	16399	3862	1931	1940	24	–	–	–	1460	–	447
iis-100-0-cov	3831	100	29	29	0	29	29	29	0	29	0
iis-bupa-cov	4803	345	36	38	0	36	36	36	0	36	0
iis-pima-cov	7201	768	33	46	0	33	33	33	0	33	0
macrophage	3164	2260	374	518	0	374	409	421	0	419	0
mspp16	561657	29280	363	363	16	–	–	358	17	477	8
n3div36	4484	22120	130800	480400	0	135000	131000	136400	0	134600	0
n3seq24	6044	119856	52200	841200	0	52260400	69000	77600	0	70200	0
neos-1440225	330	1285	36	36	6	36	–	36	0	–	0
neos-1616732	1999	200	159	160	0	160	159	159	0	159	0
neos-1620770	9296	792	9	13	1	9	9	9	0	9	0
neos-631710	169576	167056	203	555	0	215	435	390	0	398	0
neos-849702	1041	1737	0	0	16	–	–	–	–	–	–
neos-941313	13189	167910	9361	19010	10	–	21087	20459	2811	19111	0
neos18	11402	3312	16	22	0	16	17	17	0	17	0
neos808444	18329	19846	0	0	38	–	–	0	0	0	0
netdiversion	119589	129180	242	5714768	20	4900438	–	–	245598363	–	1699
ns1663818	172017	124626	86	299	1924	–	–	–	71455	–	440
ns1688347	4191	2685	27	16	6	27	–	105	104	1	2
ns1696083	11063	7982	45	53	4	–	–	301	274	10	4
ns894244	12129	21856	15	16	15	–	–	172	156	16	56
ns894788	2279	3463	7	7	7	–	–	15	8	7	3
seymour	4944	1372	423	439	0	427	424	424	0	424	0
tanglegram1	68342	34759	5182	7516	0	5431	–	–	54	–	415
tanglegram2	8980	4714	443	2010	0	443	443	443	0	445	0
toll-like	4408	2883	610	987	0	621	658	671	0	657	0
vpphard	47280	51471	5	328	5	–	205	44	36	15	12
vpphard2	198450	199999	81	718	33	–	881	177	166	133	93
wnq-n100-mw99-14	656900	10000	259	4476	0	–	2196	1893	0	2201	0

Table 8: Comparison with another General-Purpose Local Search Solver developed by a previous student at The University of Southern Denmark

9 Appendix A

9.1 Variables

While it is possible to declare the type of variables in the MPS format when declaring the bound of the variable, CoinMpsIO does only support distinction between integer and continuous variables. Since all variables of the IMADA Timetabling Problem are either binary or integer, it is assumed that a given variable x is binary if $0 \leq x \leq 1$, and integer otherwise.

9.2 Objective function

The objective function is made by going through the n coefficients c_j, $j = 1 \dots n$ of the objective function Z, that is,

$$Z = \sum_{j=1}^{n} c_j x_j = \vec{c}^T \vec{x}.$$

9.3 Constraints

The m constraints are constructed as linear expressions k_i before they are added to the model by first reading the matrix A containing the coefficients of the constraints, and then adding the bounds separately. The bounds have to be added separately because of how the MPS format is defined. This will be explained shortly.

The matrix A is of size $m \times n$ and contains the coefficients of the constraints, thus every constraint k_i is made of n coefficients $\forall i \in \{1 \dots m\}$.

By looping through each list of coefficients A_i, the expression k_i is initialized as a sum, and the expression $a_{ij}x_j$ is added to k_i for each coefficient in A_i.

$$k_i = \sum_{j=1}^{m} a_{ij} x_j \qquad \forall i \in \{1 \dots m\}$$

There is 4 types of bounds which are handled by a loop containing a switch statement. The 4 types of bounds are given by the letter in the MPS format:

E an equality constraint $A\vec{x} = \vec{b}$

L meaning a non-strict inequality constraint of type $A\vec{x} \leq \vec{b}$

G meaning a non-strict inequality constraint of type $A\vec{x} \geq \vec{b}$

R meaning a range $\vec{l} \leq A\vec{x} \leq \vec{u}$ where $\vec{l}$ is a vector of lower bounds and $\vec{u}$ is a vector of upper bounds.

For the m constraints, the bounds $b_i \, \forall i \in \{1 \dots m\}$ are added to the model.

An example of a MPS file is shown below.

```
...
ROWS
 L  LIM1
...
COLUMNS
    X         LIM1                1
    Y         LIM1                2
...
RHS
    RHS1      LIM1                9
...
```

Considering the constraint `LIM1`, the `ROWS` section defines the constraint to be of type

$$A\vec{x} \leq \vec{b}$$

The `COLUMNS` sections defines the coefficients of the constraint for the variables `X` and `Y` (1 and 2 respectively), and `RHS` defines the bound to be 9. This means that the constraint will be defined as:

$$1x + 2y \leq 9$$

However, the definition of constraint `LIM1` is divided into the different sections of the MPS file. Thus, because the MPS file contains information about each constraint multiple places in the MPS file, they have to be assembled before they are added to the model.