$\sum_x$-Optimal Solutions in Highly Symmetric Multi-Objective Timetabling Problems

Maxime Clement · Tenda Okimoto · Katsumi Inoue · Mutsunori Banbara

Abstract Multi-objective optimization plays a very important role in the real-world timetable generation. Due to its exponential nature, traditional approaches rely on scalarization methods like weighted-sum. Using a set of weights W_x, a weighted-sum allows to find one solution we define as $\sum_x$-optimal. While such method guarantees to find a Pareto-optimal solution, it cannot provide a set of solutions with various trade-offs of objectives, which can be very valuable in highly symmetric course timetabling.

We here present a scalarization-based approach to finding set of $\sum_x$-optimal solutions for multi-objective Curriculum-Based Course Timetabling (CB-CTT), one of the most widely studied course timetabling problems. Our prototype system reads a CB-CTT instance and then enumerates all solutions of vector format that have a minimal sum of penalty cost, which can subsequently be used to select various interesting solutions from a multi-objective viewpoint. The major feature of our approach is that it can guarantee not only the minimality but also Pareto-optimality. We establish the effectiveness of our approach by empirically showing the set of $\sum_x$-optimal solutions obtained for some CB-CTT instances used in the second international timetabling competition (ITC-2007).

Keywords University Course Timetabling · Multi-objective · Utilitarianism · ASP

This work was supported in part by • JSPS KAKENHI 26540122 • JSPS KAKENHI 15K00099 • JSPS KAKENHI 16H02803.

Maxime Clement[12] · Tenda Okimoto[4] · Katsumi Inoue[123] · Mutsunori Banbara[5]

[1]Principles of Informatics Research Division, National Institute of Informatics Tokyo, Japan
[2]Department of Informatics, School of Multidisciplinary Sciences, SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan
[3]Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, Japan
[4]Graduate School of Maritime Science, Kobe University, Kobe, Japan
[5]Information Science and Technology Center, Kobe University, Kobe, Japan

E-mail: maxime-clement@nii.ac.jp, tenda@maritime.kobe-u.ac.jp, inoue@nii.ac.jp, banbara@kobe-u.ac.jp

1 Introduction

The *university course timetabling* problem [14, 16, 6] is one of the representative application problems in operations research (OR) and artificial intelligence (AI) which can be generally defined as the task of assigning a number of lectures to a limited set of time slots and rooms, subject to a given set of hard and soft constraints. This problem is formalized as a combinatorial constraint optimization problem where the aim is to find an assignment of values to variables so that all hard-constraints are satisfied and the sum of all violated costs of soft-constraints is minimized.

Many real-world university course timetabling problems involve multiple criteria that can be considered separately but should be optimized simultaneously. In a multi-objective course timetabling, there usually exists an exponential number of acceptable solutions called Pareto-optimal solutions. An assignment is considered Pareto-optimal if there does not exist another assignment that is better for all objectives. Complete approaches computing the full set of Pareto-optimal solutions do not scale well for complex problems such as timetabling. A common compromise is to use a scalarization method, which is the simplest and the most widely used method to find Pareto-optimal solutions in a multi-objective timetabling problem. Such scalarization methods effectively transform a problem with multiple objectives into a mono-objective problem. The optimal solution obtained is then guaranteed to be Pareto-optimal and to satisfy an utilitarian view where the sum of objectives is optimized. Various sophisticated approximation algorithms have also been developed for finding feasible solutions, e.g., simulated annealing [10], multi-phase tabu search [6] and multi-objective evolutionary algorithm [20]. However, they cannot guarantee to find Pareto-optimal solutions.

The focus of our work is laid on finding a variety of Pareto-optimal solutions to the university course timetabling problem by using scalarization methods. Timetabling is a highly symmetrical problem, where the neighborhood of a valid timetable will often contain a number of other valid timetables. Because of this symmetry, there often exists a huge number of solution to a timetabling problem, several of which can optimize the weighted-sum. Assume that there exists three criteria/objectives, e.g., room capacity, minimum working days and isolated lectures, and the aim is to minimize the sum of all violated costs of these three objectives. Let 8 be the minimal value obtained by using the scalarization method, e.g., 5 for the violated cost of room capacity and 3 for that of isolated lectures. Now, what happens when there exists several solutions that have the same minimal value, e.g., 2 for first criterion, 5 for the second and 1 for the last. The existing scalarization method cannot capture all these solutions, since it terminates when one optimal solution has been found. However, from multi-objective perspective, these solutions can be very different from each other, offering various trade-offs of objectives and a partial representation of the Pareto front. As far as we are aware, there is no other work that focuses on providing a set of optimal solutions for university course timetabling.

In this paper, we propose a new approach for multi-objective timetabling problems. By treating the violations of each soft-constraint as an objective, we can associate a vector of penalties to any timetable. Then, we propose to compute the set of all utilitarian vectors, offering different trade-offs of objectives while minimizing their

sum. We call this set the $\sum_x$-optimal front. Solutions in this set are guaranteed to be Pareto-optimal as well as satisfy utilitarianism, i.e., the sum of the violated costs is minimal. However, as we will show in this paper, the $\sum_x$-optimal front can be quite large on some problem instances. Because of the complexity of university timetables, directly providing too many solutions to the decision makers can be of little interest. Therefore we also introduce additional criteria to extract specific solutions from the $\sum_x$-optimal front in order to provide an interesting set of solutions.

In the experiments,we use Answer Set Programming (ASP) [13, 18, 2, 11] to compute the complete $\sum_x$-optimal front of some popular university course timetabling instances used in the ITC-2007 competition [8]. ASP is an approach to declarative problem solving, combining a rich yet simple modeling language with high-performance solving capacities. It is well suited for implementing our approach as it was already used to compute some optimal solutions for CB-CTT in a previous work [1]. Using ASP, we are able to compute solutions of many of the benchmark instances and we show that many trade-offs are indeed available, validating the strength of our approach.

The rest of the paper is organized as follows. In the next section, the university course timetabling is introduced. Afterwards, some multi-objective concepts are defined for timetabling and a novel solution criterion called $\sum_x$-optimality is defined. Next, we discuss additional criteria we can use to isolate interesting solutions from the $\sum_x$-optimal front. We then experiment our novel approach with university course timetabling benchmarks from ITC-2007. Just before the concluding section, some related works are discussed.

2 CB-CTT

In this section, we describe the model of Curriculum-Based Course Timetabling [3] (CB-CTT) as it was defined for the ITC-2007 competition [15].

CB-CTT models the course timetabling problem that arises in many universities. In this model, we have a set of curricula that predefines the sets of courses a student can follow. Then, each course is made of several lectures that will take place every week. One of the simplification made compared to other models of timetabling is the omission of student sectioning, where each student has to be assigned to individual sections of a course. Even with this simplification, CB-CTT is a NP-hard problem.

In this model, we are given sets of:

- *time periods*: the time horizon is divided into days and time slots per day. A time period is a pair (day, time slot).
- *courses*: each course consists of a given number of lectures, is taught by a teacher and is attended by a given number of students.
- *curricula*: a curriculum is a set of courses.
- *rooms*: each room has a maximum capacity.

CB-CTT then consists in finding an assignment of course lectures to rooms while satisfying a set of hard-constraints:

- H_1. **Lectures**: all lectures of each course must be scheduled and they must be assigned to distinct time slots.
- H_2. **Conflicts**: lectures of courses in the same curriculum or taught by the same teacher must be all scheduled in different time slots.
- H_3. **Room Occupancy**: two lectures can not take place in the same room in the same time slot.
- H_4. **Availability**: if the teacher of the course is not available to teach that course at a given time slot, then no lecture of the course can be scheduled at that time slot.

Definition 1 (Valid timetable) A timetable T is valid if it satisfies all hard-constraints.

To be able to compare different valid timetables, other constraints qualified as soft-constraints, are used as quality measures:

- S_1. **Room Capacity**: For each lecture, the number of students that attend the course must be less than or equal the number of seats of all the rooms that host its lectures. The penalties, reflecting the number of students above the capacity, are imposed on each violation.
- S_2. **Minimum Working Days**: The lectures of each course must be spread into a given minimum number of days. The penalties, reflecting the number of days below the minimum, are imposed on each violation.
- S_3. **Isolated Lectures**: Lectures belonging to a curriculum should be adjacent to each other in consecutive timeslots. For a given curriculum we account for a violation every time there is one lecture not adjacent to any other lecture within the same day. Each isolated lecture in a curriculum counts as one violation.
- S_4. **Windows**: Lectures belonging to a curriculum should not have time windows (periods without teaching) between them. For a given curriculum we account for a violation every time there is one window between two lectures within the same day. The penalties, reflecting the length in periods of time window, are imposed on each violation.
- S_5. **Room Stability**: All lectures of a course should be given in the same room. The penalties, reflecting the number of distinct rooms but the first, are imposed on each violation.
- S_6. **Student MinMax Load**: For each curriculum the number of daily lectures should be within a given range. The penalties, reflecting the number of lectures below the minimum or above the maximum, are imposed on each violation.
- S_7. **Travel Distance**: Students should have the time to move from one building to another one between two lectures. For a given curriculum we account for a penalty every time there is an instantaneous move: two lectures in rooms located in different building in two adjacent periods within the same day. Each instantaneous move in a curriculum counts as one penalty.
- S_8. **Room Suitability**: Some rooms may not be suitable for a given course because of the absence of necessary equipment. Each lecture of a course in an unsuitable room counts as one penalty.
- S_9. **Double Lectures**: Some courses require that lectures in the same day are grouped together (double lectures). For a course that requires grouped lectures,

Table 1 Formulations of CB-CTT

Constraint	UD_1 [9]	UD_2 [8]	UD_3 [4]	UD_4 [4]	UD_5 [4]
H_1 : Lectures	H	H	H	H	H
H_2 : Conflicts	H	H	H	H	H
H_3 : Room Occupancy	H	H	H	H	H
H_4 : Availability	H	H	H	H	H
S_1 : Room Capacity	1	1	1	1	1
S_2 : Minimum Working Days	5	5	0	1	5
S_3 : Isolated Lectures	1	2	0	0	1
S_4 : Windows	0	0	4	1	2
S_5 : Room Stability	0	1	0	0	0
S_6 : Student MinMax Load	0	0	2	1	2
S_7 : Travel Distance	0	0	0	0	2
S_8 : Room Suitability	0	0	3	H	0
S_9 : Double Lectures	0	0	0	1	0

every time there is more than one lecture in one day, a lecture non-grouped to another is not allowed. Two lectures are grouped if they are adjacent and in the same room. Each non-grouped lecture counts as one penalty.

Using those soft-constraints, we can represent the quality of a timetable T as a vector of penalties $V(T) = (v_1, v_2, \ldots, v_9)$ where v_i is the number of penalties for constraint S_i $(1 \leq i \leq 9)$. Then, if we know the preferences of the decision makers, we can associate a weight to each soft-constraint such that the lower the weight, the lower the importance of the constraint. A weight of 0 corresponds to completely ignoring the constraint.

Various formulations of the problem uses different weights, as seen in Table 1. In the table, lines represent constraints and columns represent the various formulations (UD_1 to UD_5). A "H" in a cell indicates that the formulation uses the constraint as a hard-constraint. A number in a cell indicates the weight associated with the soft-constraint. UD_1 is the first set of weights proposed for the CB-CTT problem [9] and only considers three soft-constraints. UD_2 is the set of weights proposed for the ITC-2007 competition [8], where they extend UD_1 with one additional soft-constraint. UD_3, UD_4, and UD_5 [4], were proposed in a latter work in order to offer variations of the original problem by considering 4 new soft-constraints. Those five formulations are the most commonly studied and are the ones we will use throughout this paper. In real applications however, those formulations might not be directly used since different universities might want to consider different constraints and weights.

Using a formulation UD_x with the associated set of weights W_x, we can transform a vector a penalties $V(T)$ into a weighted vector $V_x(T)$.

Definition 2 (Weighted-Vector) Given a timetable T and a set of weights W_x corresponding to formulation UD_x, we have the corresponding weighted-vector $V_x(T) = (v_1 \times w_1, v_2 \times w_2, \ldots, v_9 \times w_9)$ with $v_i \in V(T)$.

The most common approach to the CB-CTT problem aims at finding a solution such that the sum of values in its corresponding weighted-vector is minimized. We call such solution a $\sum_x$-optimal solution and it is the typical result to scalarization methods.

Definition 3 ($\sum_x$-Optimal solution) Given a valid timetable T and a formulation UD_x, we say T is $\sum_x$-optimal if it minimizes

$$\sum_{v_i \in V_x(T)} v_i$$

Most existing works on timetabling stop at this stage, after finding one solution for a given set of weights. However, due to the highly symmetrical nature of timetabling problems, there might exists several $\sum_x$-optimal solution. In the next section, we propose to find more than one solution by considering a subset of all the $\sum_x$-optimal solutions.

3 Set of Solutions

In the previous section, we showed how to evaluate the quality of a timetable when preferences over the soft-constraints are known. Each timetable can be associated with a single value and most works on timetabling will find one timetable minimizing this value.

There are two issues with this approach, (i) it only provides the decision makers with one possibility and (ii) it overlooks the various trade-offs of penalties available. In timetabling and many other decision problems, two solutions that might appear equivalent for a given criteria (same weighted-sum) might actually be very different in the eyes of the decision makers. For example, there might exists three vectors $\{9, 1\}$, $\{5, 5\}$ and $\{1, 9\}$ that share the same sum (10) but with very different trade-offs of objectives.

In this section, we will consider how to provide more than one solution to the Curriculum-Based Course Timetabling (CB-CTT) problem by improving upon the weighted-sum criterion.

3.1 Pareto Front

For a given CB-CTT, each timetable can be associated with a weighted-vector of penalties. By considering each penalty as an objective to optimize, we can introduce the concept of Pareto-optimality for timetables.

Definition 4 (Pareto Dominance) Given two vectors V and V', we say that *V Pareto dominates V'*, denoted by $V \succ V'$, iff V is partially less than V', i.e., (i) it holds $v_i \in V \leq v'_i \in V'$ for all i, and (ii) there exists at least one i such that $v_i \in V < v'_i \in V'$

Definition 5 (Pareto-Optimal Solution) A timetable T is Pareto-optimal for UD_x if there does not exist another timetable T' such that $V_x(T) \prec V_x(T')$.

The penalty vectors corresponding to those Pareto-optimal timetables form what is called the Pareto Front.

Definition 6 (Pareto Front) The Pareto Front is the set of vectors we can obtain from all Pareto-optimal solutions.

Table 2 Example of Solutions

	Penalties			
	S_1	S_2	S_3	S_5
T_1	5	0	3	2
T_2	6	1	0	3
T_3	7	1	1	5
T_4	2	1	2	2
T_5	5	0	3	5

By finding one solution for each possible vector of the Pareto front, we can provide every interesting trade-off of penalties. However, finding the Pareto front of complex problems is often too difficult. In the worst case, each assignment can be Pareto-optimal and with large problems such as CB-CTT, we can have thousands of variables to assign to thousands of pair room/period. The number of solutions being exponential, using complete methods becomes unrealistic on large problem instances. Because of this complexity, it is important to consider techniques that can quickly provide an approximation of the Pareto front. We thus propose to find a subset of the Pareto front while still complying with the weighted-sum criterion.

3.2 $\sum_x$-Optimal Front

Since the weighted-sum is the state of the art approach for CB-CTT, we can assume that a timetable that is $\sum_x$-optimal is a good solution. However, there might exists more than one such solution, potentially offering very different trade-offs between penalties. Finding the set of all $\sum_x$-optimal solutions would allow to offer a set of good timetables to choose from. The user can then select one of the possible choices by considering his preferred trade-off of objectives, or even criteria that were not originally formulated in the problem. Since those solutions minimize the weighted-sum of the penalties, they are all Pareto-optimal, meaning that there exists no other timetable strictly better for all objectives.

Property 1 (Pareto-Optimality of $\sum_x$-Optimal Solutions) A vector V that is $\sum_x$-optimal is also Pareto-optimal for UD_x.

Due to the size and high symmetry of CB-CTT problems, there might exists a huge number of timetables that share the same penalty vector. In our work, we are focusing on providing various penalty trade-offs and do not care about providing more than one timetable for each possible vector. Thus, we will focus on the subset of the Pareto front we can find using $\sum_x$-optimal solutions.

Definition 7 ($\sum_x$-Optimal Front) The $\sum_x$-optimal front is the set of vectors that can be obtained by $\sum_x$-optimal solutions.

3.3 Example

Let us now review all the concepts introduced in this section using a simple example.

Example 1 (CB-CTT) Let us consider an example of CB-CTT that accepts five valid timetables T_1, T_2, T_3, T_4 and T_5 whose penalty vectors are shown in Table 2, where column S_i contains the penalty for soft-constraint S_i and line T_i contains the vector $V(T_i)$.

First, let us consider formulation UD_2. We consider the four objectives S_1, S_2, S_3 and S_5 with weights 1, 5, 2 and 1 respectively. By applying the set of weight to each vector, we obtain the following weighted vectors: $V_2(T_1) = \{5, 0, 6, 2\}$, $V_2(T_2) = \{6, 5, 0, 3\}$, $V_2(T_3) = \{7, 5, 2, 5\}$, $V_2(T_4) = \{2, 5, 4, 2\}$ and $V_2(T_5) = \{5, 0, 6, 5\}$.

Based on the definition of Pareto-optimality, this problem has three Pareto-optimal timetables when using UD_2: T_1, T_2 and T_4. T_3 is not Pareto-optimal since it is dominated by T_2 ($\{6, 5, 0, 3\} \succ \{7, 5, 2, 5\}$) and T_5 is dominated by T_1 ($\{5, 0, 6, 2\} \succ \{5, 0, 6, 5\}$).

The Pareto front for this problem is the set of vectors obtained from Pareto-optimal solutions: $\{\{5, 0, 6, 2\}, \{6, 5, 0, 3\}, \{2, 5, 4, 2\}\}$.

Now if we consider the weighted sum of each vector, we can find two $\sum_2$-optimal solutions, T_1 and T_4. Those timetables minimize the sum of weighted penalties with a sum equal to 13 ($5 + 0 + 6 + 2 = 13$ for T_1 and $2 + 5 + 4 + 2 = 13$ for T_4). T_2 is not $\sum_2$-optimal since its corresponding sum is not minimal($6 + 5 + 0 + 3 = 14$).

We can then say that the $\sum_2$-optimal front of this problem is $\{\{5, 0, 6, 2\}, \{2, 5, 4, 2\}\}$.

Let us now consider formulation UD_1. This time we only consider three objectives S_1, S_2, S_3 with weights 1, 5, 1 respectively. We obtain the following weighted vectors: $V_1(T_1) = \{5, 0, 3\}$, $V_1(T_2) = \{6, 5, 0\}$, $V_1(T_3) = \{7, 5, 1\}$, $V_1(T_4) = \{2, 5, 2\}$ and $V_1(T_5) = \{5, 0, 3\}$.

We now have four Pareto-optimal timetables: T_1, T_2, T_4 and T_5.Only T_3 is not Pareto-optimal since it is still dominated by T_2 ($\{6, 5, 0\} \succ \{7, 5, 1\}$).

The Pareto front for this problem is the set of vectors obtained from Pareto-optimal solutions: $\{\{5, 0, 3\}, \{6, 5, 0\}, \{2, 5, 2\}\}$.

Now if we consider the weighted sum of each vector, we can find two $\sum_1$-optimal solutions, T_1 and T_5. Those two timetables minimize the sum of weighted penalties with a sum equal to 8 ($5 + 0 + 3 = 8$). T_2 and T_4 are not $\sum_1$-optimal since their corresponding sum are not minimal ($6 + 5 + 0 = 11$ for T_2 and $2 + 5 + 2 = 9$ for T_4).

When we consider the $\sum_1$-optimal front, we only care about the vectors and not the assignment (timetable). Since here both T_1 and T_5 share the same vector, we have the $\sum_1$-optimal front $\{\{5, 0, 3\}\}$.

Next, we will consider additional criteria that we can use to identify solutions from the $\sum_x$-optimal front with interesting properties.

4 Subset of the $\sum_x$-Optimal Front

With the $\sum_x$-optimal front, we can provide a set of solutions to a decision maker. While we can expect this set to be much smaller than the Pareto front, its size might still prove too large to handle for a human. A typical university timetable being made of thousands of lectures assigned to thousands of rooms and periods, if a human wants a good understanding of the different timetables, it is imperative to present him with a limited set of solutions. In the case of multiple decision makers, proposing

a large number of alternatives can also results in each decision maker preferring a different timetable. Thus we believe it is important to consider additional criteria when proposing a set of solutions.

The solutions provided by a $\sum_x$-optimal front already satisfy an utilitarian criterion [22], guaranteeing a minimal sum of penalties. We will now propose two additional criteria we can consider to isolate specific solutions from the $\sum_x$-optimal front.

4.1 Egalitarianism

With egalitarianism, instead of focusing exclusively on the weighted-sum, the goal is to obtain a balanced solution where penalties are spread as much as possible between the objectives (soft-constraints). To find the most egalitarian solutions from the $\sum_x$-optimal front, we use a lexicographic ordering [22]. Basically, we want to find the largest minimum value. This requires to arrange a vector V in an increasing order, denoted $V^<$. We then define the following ordering:

Definition 8 (Lexicographic ordering) Given two vectors V and V' of size m, we say that V *lexicography precedes* V', denoted by $V \prec^{lex} V'$, iff there is a $i \in \{1, \ldots, m\}$ such that $V_i^< < V_i'^<$ and, if $i > 1$, then $\forall j \in \{1, \ldots, i-1\}, V_j^< = V_j'^<$.

The most egalitarian vector from the $\sum_x$-optimal front is the one that is preceded by all others in the lexicographic ordering, which tends to maximize the minimum value of its vector. With CB-CTT, this solution will tend to have the most balanced penalties, which can be interesting if many small constraint violations are considered better than a few constraints being violated numerous times. For other models of the timetabling problems, egalitarianism can be used to find fair solutions [17], for example to have penalties spread between different school departments, different teachers, different classes, ...

Example 2 (Egalitarianism) Let us consider the same example as Example 1 with the problem shown in Table 2. Without applying any weight, we can order the five vectors of penalty using the lexicographic ordering. First, let us reorder each vector $V(T_i)$ into $V^<(T_i)$. We obtain $V^<(T_1) = \{0, 2, 3, 5\}$, $V^<(T_2) = \{0, 1, 3, 6\}$, $V^<(T_3) = \{1, 1, 5, 7\}$, $V^<(T_4) = \{1, 2, 2, 2\}$ and $V^<(T_5) = \{0, 3, 5, 5\}$. Here, $V(T_2)$ lexicography precedes all other vectors. Its first value (corresponding to the minimum) in $V^<(T_2)$ is 0, which is inferior to the first value in $V^<(T_3)$ and $V^<(T_4)$ ($0 < 1$), thus $V(T_2) \prec^{lex} V(T_3)$ and $V(T_2) \prec^{lex} V(T_4)$. Its first value is equal to the first value in $V^<(T_1)$ and $V^<(T_5)$, but then the next value (1) is inferior (2 for T_1 and 3 for T_5), thus we have $V(T_2) \prec^{lex} V(T_1)$ and $V(T_2) \prec^{lex} V(T_5)$. We can say that T_2 is the less balanced of the solutions.

The most balanced solution, said to be egalitarian, is T_4 as it is lexicography preceded by all the other vectors. Its minimum value of 1 is larger than the minimum value of T_1, T_2 and T_5 and we can write $V(T_1) \prec^{lex} V(T_4)$, $V(T_2) \prec^{lex} V(T_4)$ and $V(T_5) \prec^{lex} V(T_4)$. Its minimum value of 1 is the same as T_3, but then the next value (2) is larger than in T_3 (1), thus $V(T_3) \prec^{lex} V(T_4)$. We can say that T_4 is the most balanced vector and is the egalitarian solution here.

4.2 Constraint Satisfaction

Another aspect we can consider is the maximization of the number of completely satisfied soft-constraints. No violation of some objectives can be a good selling point for a decision maker and it is easy to explain the different with another timetable that does not satisfy the same constraints.

This focus on satisfying as many soft-constraints as possible could be simply expressed in the original problem. However, if we focus on the satisfaction of the objectives, instead of their optimization, we can end up with bad solutions that have a huge amount of violations on a few objectives.

Here, since we work from the $\sum_x$-optimal front, we always have a minimal sum of weighted penalties.

In the next section, we will present experimental results and show the interest of those criteria.

5 Experiments

In this section, we first present our method to enumerate all vectors in the $\sum_x$-optimal front before showing some results obtained on instances from the ITC-2007 competition [8].

5.1 Method

To conduct those experiments, we used an encoding of the CB-CTT with Answer set programming (ASP) that was shown to be a promising approach for timetabling [1], offering an efficient way to find one $\sum_x$-optimal solution of a problem.

ASP is a form of declarative programming mostly used to solve NP-hard problems. An ASP problem is made up of rules that represents conditions to be satisfied and facts that are known to be true. An ASP solver will encode such problem into a logic program before searching for some stable models, corresponding to solutions of the original problem.

We represent the CB-CTT problem as a set of rules and a problem instance is a set a set of fact such as the number of days, period per days, courses, teachers, ... We can then search for a timetable that satisfies all the rules from the hard-constraints and that minimizes the sum of penalties from the soft-constraints.

While in theory it is possible to enumerate all valid timetables, in practice, due to the symmetry of CB-CTT, it is too complex and the enumeration takes too much time. Thus we focus on the vector of penalties and only find one corresponding timetable.

In the method we used, we find $\sum_x$-optimal solutions one by one. For each newly found vector of penalty, we extend the original problem by adding a rule forbidding the same vector to be a solution. This method ends when there exists no more solution to our extended problem, meaning that we found the complete $\sum_x$-optimal front.

Because the time to find a new solution can be very long, we can note that it is possible to stop at anytime during our method and then use the $\sum_x$-optimal solutions

Table 3 $\sum_4$-optimal front for comp17 (with optimal weighted sum of 21)

#	S_1	S_2	S_4	S_6	S_9	#	S_1	S_2	S_4	S_6	S_9
1	0	13	0	8	0	12	0	12	0	9	0
2	0	11	3	7	0	13	0	11	0	10	0
3	0	12	2	7	0	14	0	12	3	6	0
4	0	11	1	9	0	15	0	9	4	8	0
5	0	12	1	8	0	16	0	10	4	7	0
6	0	11	2	8	0	17	0	11	4	6	0
7	0	10	3	8	0	18	0	14	1	6	0
8	0	10	2	9	0	19	0	14	0	7	0
9	0	9	3	9	0	20	0	15	0	6	0
10	0	13	2	6	0	21	0	10	1	10	0
11	0	13	1	7	0						

found so far. In the best case, we stopped during the last step which is only used to prove that we indeed found the complete $\sum_x$-optimal front. Else, we only have an incomplete $\sum_x$-optimal front that can still be used and might still provide a number of interesting trade-offs.

5.2 Results

We now present results obtained by using our method on some instances proposed for the International Timetabling Competition of 2007 [8]. There is a total of 21 instances of various complexity. The solving of some of those instances are still open with any $\sum_x$-optimal solutions yet to be found.

Table 3 shows the $\sum_4$-optimal solutions of instance *comp17*. For the formulation UD_4, *comp17* is a difficult instance that had no known $\sum_4$-optimal solutions until now. As we can see, this instance has 21 $\sum_4$-optimal solutions, minimizing the sum of penalties at 21. While soft-constraints S_1 and S_9 are always fully satisfied, we can observe different trade-offs over the 3 other objectives. S_2 varies between 9 and 15, S_4 varies between 0 and 4, and S_6 varies between 6 and 10. Those different trade-offs can be very interesting to a decision maker and we can notice some clear differences between the available vectors. Most notably, some vectors can completely satisfy the constraint S_4. As we discussed in the previous section, it can sometimes be preferable to completely satisfy one more objective at the cost of increasing the penalty of two others (vector #1 with penalties $\{0, 13, 0, 8\}$ for example). We also point out the vector #15 with penalties $\{0, 9, 4, 8, 0\}$, which is the most egalitarian vector, offering a good balance between S_2, S_4 and S_6.

Table 4 shows the $\sum_4$-optimal solutions of instance *comp04*. As we can see, this instance has 13 $\sum_4$-optimal solutions, minimizing the sum of penalties at 13. Like in *comp17*, soft-constraints S_1 and S_9 are always fully satisfied. However, the variations of the three others objectives are quite different, especially S_4 which only varies between a penalty of 0 and 1. We can thus notice that for each vector where $S_4 = 1$, we can find a very close vector where S_4 is completely satisfied (vector #2 $\{0, 9, 1, 3, 0\}$ and #1 $\{0, 10, 0, 3, 0\}$ for example). If we value the complete satisfaction of the objectives, we can consider ignoring vectors that violates S_4 but are very close to vectors

Table 4 $\sum_4$-optimal front for comp04 (with optimal weighted sum of 13)

#	S_1	S_2	S_4	S_6	S_9	#	S_1	S_2	S_4	S_6	S_9
1	0	10	0	3	0	8	0	8	1	4	0
2	0	9	1	3	0	9	0	11	0	2	0
3	0	8	0	5	0	10	0	6	0	7	0
4	0	9	0	4	0	11	0	12	0	1	0
5	0	7	0	6	0	12	0	11	1	1	0
6	0	10	1	2	0	13	0	6	1	6	0
7	0	7	1	5	0						

where $S_4 = 0$. However, if we care more about having a good balance between the penalties, the best solution here would be the vector #13 $\{0, 6, 1, 6, 0\}$. This vector, while providing the most balance, still greatly favors S_2 and S_6 compared to S_4, showing that we cannot always find a perfectly-balanced solution.

Previous approaches that focus on finding one solution usually produce only one of the vectors we show here. In light of the many possible trade-offs and the clear differences between two $\sum_x$-optimal solutions, we showed the importance to present those alternatives to a decision maker.

Table 5 shows the runtime, the size of the $\sum_x$-optimal front as well as the optimal sum of penalties for some instances and formulations. Due to the very long time it takes to compute the $\sum_x$-optimal front, we have yet to produce results for all possible instances and formulations and for this experiment, we limited the time to find a new solution (or prove unsatisfiability) to 6 hours. Each line in the table represent data for a given instance and formulation. The size represent the size of the $\sum_x$-optimal fronts and sum represents the sum of penalties for the vectors found. Total time represents the total CPU time it took to find the complete $\sum_x$-optimal front. Unsat time is the duration of the last step of our method, which requires to prove that with the added constraints, the problem is no longer satisfiable, meaning that we found the complete $\sum_x$-optimal front.

While we were able to solve many instances using our method, we were only able to completely find $\sum_x$-optimal front for the formulation UD_4. Because UD_4 uses an additional hard-constraint, the number of valid timetables is greatly reduced compared to other formulations, making it possible to find a new $\sum_4$-optimal solution within the 6 hours limit.

For other formulations, we are only able to solve instances with a unique $\sum_x$-optimal solution.

Regarding the runtime, it will greatly vary based on the complexity of the instance and the formulation used. Additionally, since we solve the same problem several times, adding a constraint after each newly found vector, we can expect that the higher the size of the $\sum_x$-optimal front, the longer the time it takes to completely find it. It results that some instance were solved in less than one second (*comp11* for UD_1, UD_2 and UD_3) while others took up to 30 minutes (*comp10* for UD_2) and even around 94 hours (*comp17* for UD_2 and UD_4). We can see from the unsat time column that for some instances, the majority of the time is taken to prove that there does not exist anymore vector belonging to the $\sum_x$-optimal front (*comp04* for UD_5 for example). While having the complete $\sum_x$-optimal front can be important, it can also be ignored

Table 5 Results for finding the $\sum_x$-optimal front on various instances

instance	formulation	size	total time (s)	unsat time (s)	sum
comp04	UD3	1	1.8	0.92	2
comp04	UD4	13	764.02	593.66	13
comp04	UD5	1	12705.82	12664.52	49
comp06	UD2	1	1290.64	1102.6	27
comp06	UD3	1	7.26	3.78	8
comp07	UD1	1	48.22	9.72	3
comp07	UD2	1	18206.52	11129.4	6
comp07	UD3	1	214.38	0.98	0
comp07	UD4	4	19966.38	241.94	3
comp08	UD3	1	16.18	7.12	2
comp08	UD4	14	2316.62	1343.14	15
comp08	UD5	1	2610.72	2524.4	55
comp10	UD1	1	2.8	1.42	2
comp10	UD2	1	1565.24	925.84	4
comp10	UD3	1	2.96	0.8	0
comp10	UD4	6	106.48	12.4	3
comp11	UD1	1	0.28	0.12	0
comp11	UD2	1	0.4	0.14	0
comp11	UD3	1	0.72	0.32	0
comp11	UD4	1	3.38	1.08	0
comp11	UD5	1	838.66	0.58	0
comp14	UD3	1	1.86	0.58	0
comp14	UD4	2	55.04	31.4	14
comp16	UD1	1	4207.06	4203.16	11
comp16	UD2	1	48.04	13.8	18
comp16	UD3	1	4.18	2.56	4
comp16	UD4	3	127.5	60.96	7
comp17	UD3	1	6.8	3.78	12
comp17	UD4	21	341,681.21	42,963.86	21
comp18	UD3	1	1.14	0.44	0
comp20	UD1	1	767.78	389.74	2
comp20	UD2	1	763.46	210.4	4
comp20	UD3	1	371.94	0.92	0

if a decision needs to be taken within a short amount of time. Using our method as an anytime approach, we can provide a set of $\sum_x$-optimal solutions more quickly but at the cost of potential incompleteness.

5.3 Results with neutral weights

Choosing weights for each soft-constraint can be a difficult task for a decision maker, and the impact of those weights is hard to estimate in advance. Thus, it might sometimes be better to start by considering the neutral case where no preferences are given and where the decision maker simply needs to provide the constraints to take into account. We evaluated this case and ran experiments where we put the weights of the considered soft-constraints to 1, showing the results in Table 6. Because UD_4 already used weights of 1, we focus on the other formulations and are able to find interesting $\sum_x$-optimal fronts on a few instances.

Table 6 Results for finding the $\sum_x$-optimal front on various instances (using weights of 1)

instance	formulation	size	total time (s)	unsat time (s)	sum
comp04	UD1	8	84,216.68	83,929.64	12
comp04	UD2	8	71,468.52	71,434.84	12
comp04	UD3	1	2.04	1.00	1
comp04	UD5	1	25.64	13.28	13
comp07	UD1	4	168.46	30.52	3
comp10	UD1	2	3.38	1.41	2
comp10	UD2	2	1,883.54	552.79	2
comp10	UD3	1	2.94	0.82	0
comp11	UD1	1	0.28	0.12	0
comp11	UD2	1	0.86	0.14	0
comp11	UD3	1	0.98	0.32	0
comp11	UD5	1	1,121.50	0.58	0
comp16	UD1	1	5.24	2.82	5
comp16	UD2	1	33.56	7.66	5
comp16	UD3	1	13.18	11.32	2
comp17	UD2	5	337,888.42	336,166.34	19
comp17	UD3	2	42.72	14.69	6

Table 7 $\sum_2$-optimal front for comp04 (with weights of 1, optimal sum of 12)

#	S_1	S_2	S_3	S_5	#	S_1	S_2	S_3	S_5
1	0	10	2	0	5	0	7	5	0
2	0	9	3	0	6	0	12	0	0
3	0	11	1	0	7	0	6	6	0
4	0	8	4	0	8	0	5	7	0

For *comp04*, we were able to find $\sum_x$-optimal front of size 8 for both UD_1 and UD_2 (with weights of 1) and we show the corresponding $\sum_2$-optimal front in Table 7 In this case, S_1 and S_5 are both completely satisfied and the 8 solutions offer different trade-offs between S_2 and S_3. Only one vector (#6 with penalties $\{0, 12, 0, 0\}$) can completely satisfy three of the soft constraints and there exists one well balanced vector (#7 with penalties $\{0, 6, 6, 0\}$).

6 Related Works

Curriculum-Based University Timetabling has been the subject of numerous publications in the last decade. As far as we are aware, no existing works have considered a subset of the Pareto front and only a few works have used multi-objective approaches. We now present some of those related works.

The first work proposes a multi-objective evolutionary algorithm for university class timetabling [7]. By optimizing two objective functions, the authors show that better results can be obtained compared to mono-objective optimization. Such approach produces trade-off solutions between different objective functions, but with no guarantee of optimality. Compared to our work, all the solutions we provide are Pareto-optimal (and also optimal for the usual weighted-sum).

The second work apply multi-objective methods to university timetabling [12] by comparing the weighted-sum to a reference point based approach, using local search algorithms. Using a reference point allows to search for a timetable as close as possible to an ideal solution. This work shows that the weighted-sum offers very unbalanced vectors and that giving a higher weight to one objective does not guarantee it will be better optimized. Compared to the weighted-sum, the reference point allows much more balanced vectors in practice but with no guarantee of obtaining the most egalitarian solution.

The third work considers fairness for CB-CTT [17]. While it does not treat each soft-constraint as a single objective, it considers trade-offs between the quality of the timetable (weighted-sum) and its fairness. The fairness is defined between different university department and the goal is to spread the constraint violations equally between the different departments. They first compute a timetable with the best quality possible, and then search for interesting trade-offs with the fairness, with fairer solutions usually being of lesser quality.

The final work uses Integer Programming to perform a lexicographic optimization of the objectives [19]. They consider each soft-constraint independently and optimize them one by one. The first solution obtained minimizes the first objective. Then, a second solution minimizes the second objective without increasing the first one. This is repeated for all objectives. This approach is quite effective to find a good timetable, but there exist a bias toward the first objectives optimized as there does not always exist another solution that improve the next objective without increasing the previous ones.

Those works use some multi-objective techniques but either do not offer a set of solutions or cannot guarantee their optimality. Our approach has the advantage to comply with the most used solution criteria for CB-CTT (minimization of the weighted-sum) while offering more than one solution to choose from.

7 Conclusion

In this work, we proposed a new approach to the Curriculum-Based Course Timetabling. Approaching the problem as a multi-objective one, we wanted to provide several timetables to choose from. Thus, we defined a subset of the Pareto Front we called $\sum_x$-optimal front. This subset contains all vectors that minimize a weighted-sum of penalties, the weighted-sum being the most used criteria for CB-CTT. In comparison, the majority of previous works using this criteria only provide one solution. We showed in our experiments however that two vectors minimizing a weighted-sum can be very different. Those differences are very important to a decision maker and should not be ignored. While this paper focuses on CB-CTT, we believe our approach can be applied to many other timetabling problems. Additionally, we proposed simple ways to isolate solutions from the $\sum_x$-optimal front. One interesting criteria is egalitarianism, where we search for a good balance between all objectives. Another consideration can be the satisfaction of as many objectives as possible. Usually, focusing on those criteria is made at the cost of the overall quality of the solution. Here,

since we first focus on utilitarianism and then consider additional criteria, we always keep the guarantee of minimizing the sum of objectives.

In future works, we want to develop more efficient methods to compute a set of solutions for timetabling problems. We plan on using asprin [5], a tool that was recently proposed for expressing preferences in Answer Set Programming. Using this tool, we could directly specify the desired set of solutions (for example the $\sum_x$-optimal front) and compute it in one shot. In addition to an increased in efficiency, a number of new criteria could easily be implemented and will be the topic of future research.

We also plan on considering other subsets of the Pareto front that do not focus on the utilitarian criterion (minimizing a weighted-sum). It can be important in some cases to consider solutions where the sum of penalties is not minimal but where the trade-offs are more interesting. For example if the objectives represent penalties with different teachers, it can be more important to have the most egalitarian solution so that no teacher can feel jealous about other teachers.

Another approach that could be interesting to study is the search for a subset of representative solutions [21]. Since the Pareto front is often too hard to compute and too large to analyze by a human, it would interesting to be able to provide a subset that best represents the complete Pareto front. While the $\sum_x$-optimal front provide a subset of solutions with various trade-offs, more extremes solutions that are not $\sum_x$-optimal could also be included in order to represents all the alternatives available. Finally, we will also apply our approach to other timetabling or scheduling problems as they often involve multiple criteria.

References

1. M. Banbara, T. Soh, N. Tamura, K. Inoue, and T. Schaub. Answer set programming as a modeling language for course timetabling. *TPLP*, 13(4-5):783–798, 2013.
2. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
3. A. Bettinelli, V. Cacchiani, R. Roberti, and P. Toth. An overview of curriculum-based course timetabling. *TOP*, 23(2):313–349, 2015.
4. A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, 194(1):59–70, 2012.
5. G. Brewka, J. P. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In *AAAI*, pages 1467–1474, 2015.
6. E. Burke, J. Marecek, A. Parkes, and H. Rudová. Decomposition, reformulation, and diving in university course timetabling. *Computers & OR*, 37(3):582–597, 2010.
7. D. Datta, K. Deb, and C. M. Fonseca. *Multi-Objective Evolutionary Algorithm for University Class Timetabling Problem*, pages 197–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
8. L. Di Gaspero, B. McCollum, and A. Schaerf. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical report, Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen's University, Belfast, United Kingdom, 2007.
9. L. Di Gaspero and A. Schaerf. Multi-neighbourhood local search with application to course timetabling. In *Practice and theory of automated timetabling IV*, pages 262–275. Springer, 2002.
10. A. R. F. Melício, P. Calderia. Solving timetabling problem with simulated annealing. In *Kluwer Academic Press*, pages 171–178, 2000.
11. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

12. M. Geiger. Multi-criteria curriculum-based course timetabling-a comparison of a weighted sum and a reference point based approach. In M. Ehrgott, C. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, volume 5467 of *Lecture Notes in Computer Science*, pages 290–304. Springer Berlin Heidelberg, 2009.
13. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
14. C. Gotlieb. The construction of class-teacher time-tables. In *IFIP Congress*, pages 73–77, 1962.
15. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Gaspero, Q. Rong, and B. Edmund. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.
16. S. MirHassani and F. Habibi. Solution approaches to the course timetabling problem. *AI Review.*, 39(2):133–149, 2013.
17. M. Mühlenthaler and R. Wanka. Fairness in academic course timetabling. In *Practice and Theory of Automated Timetabling X, tenth international conference, PATAT 2014*, pages 114–130, 2012.
18. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.
19. A. E. Phillips, H. Waterer, M. Ehrgott, and D. M. Ryan. Integer programming methods for large-scale practical classroom assignment problems. *Computers & Operations Research*, 53(0):42 – 53, 2015.
20. P. Ross, E. Hart, and D. Corne. Genetic algorithms and timetabling. *Advances in Evolutionary Computing*, pages 755–771, 2003.
21. N. Schwind, T. Okimoto, M. Clement, and K. Inoue. Representative solutions for multi-objective constraint optimization problems. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016.
22. N. Schwind, T. Okimoto, S. Konieczny, M. Wack, and K. Inoue. Utilitarian and egalitarian solutions for multi-objective constraint optimization. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 170–177, Nov 2014.