Cyclic Transfers in School Timetabling

Gerhard Post[12], Samad Ahmadi[3], and Frederik Geertsema[4⋆]

[1] Department of Applied Mathematics, University Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands
[2] ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands
[3] Department of Informatics, Faculty of Technology, De Montfort University,
The Gateway, Leicester, LE1 9BH, UK
[4] GEPRO, St. Jacobsstraat 26-28, 8911 HV Leeuwarden, The Netherlands

Abstract. In this paper we propose a neighbourhood structure based on sequential/cyclic moves and a Cyclic Transfer algorithm for the high school timetabling problem. This method enables execution of complex moves for improving an existing solution, while dealing with the challenge of exploring the neighbourhood efficiently. An improvement graph is used in which certain negative cycles correspond to the neighbours; these cycles are explored using a recursive method. We address the problem of applying large neighbourhood structure methods on problems where the cost function is not exactly the sum of independent cost functions, as it is in the set partitioning problem. For computational experiments we use four real world datasets for high school timetabling in the Netherlands and England. We present results of the cyclic transfer algorithm with different settings on these datasets. The costs decrease by 8% to 28% if we use the cyclic transfers for local optimization compared to our initial solutions. The quality of the best initial solutions are comparable to the solutions found in practice by timetablers.

1 Introduction

In the past 25 years extensive research has been carried out on automated high school timetabling. This research ranges from theoretical investigations and surveys [44,19,26,38,15,45,14,40] to case studies in specific schools from different countries (such as Australia [3,28,29], Brazil [35,36], Greece [11,43], Italy [39,18], The Netherlands, [24,46,25], Spain [9], and the UK [47]). Investigation into these case studies demonstrates that school timetabling problems change from one country to another based on different educational systems and philosophies. Design and implementation of algorithms that can deal with this variety of constraints and objectives is a great challenge. Our research on a standard modelling language for timetabling problems and on standard data formats for high school timetabling problems [33] enables us to deal with this problem in a wider context than is normally dealt within the literature.

⋆ This research has been supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

The aim of this paper is to design a neighbourhood structure and neighbourhood search method for the high school timetabling problem based on sequential moves and a cyclic transfers algorithm. Improvements to timetables are represented by certain negative cycles in an improvement graph.

The main contributions of this research are:

- Modelling the high school timetabling problem in a suitable format with appropriate definitions of moves/cyclic transfers; as shown in Section 4, the model used here is more complicated than the one used in [27,1].
- Proposing a method of resolving the problem of interdependency of different elements of the objective function for the high school timetabling problem, and showing that even an 'approximate' improvement graph can be useful in practice.
- Contributing an algorithm for finding admissible negative cycles, based on the lemma in Subsection 4.3.
- Making public real world benchmark data sets.

In the following section the common concepts and the terminology used in this paper are described. Section 3 gives a literature review, and in Section 4 we describe the improvement graph and the algorithm for searching negative cycles. Section 5 gives an overview of the datasets, and in Section 6 the computational experiments are described and the results presented. Finally Section 7 contains our major conclusions.

2 Problem definition

High school timetabling is the problem of assigning resources to lessons. The objective is to assign a fixed set of time slots and rooms to a given set of lessons subject to constraints. Assets and possible constraints considered in this paper are described in the next subsection. Constraints for the school timetabling problem fall into two categories: hard constraints and soft constraints. The hard constraints describe which assignments of time slots and rooms to lessons are regarded as feasible. The most important set of hard constraints are the 'no clash' constraints: a teacher, student or room can only be scheduled at most once in a time slot.

Soft constraints express the preferences and the quality of the timetable. The cost of a solution is measured by the weighted sum of violations of the soft constraints for every teacher and student. In this paper allocation of rooms to lessons is based on room types and not specific rooms; there are no soft constraints expressing room preferences.

A lesson is defined as a meeting between a group of students and one or more teachers at the same time and in the same venue(s). We assume that the students and teachers of a lesson are given beforehand. The problem of finding a timetable for lessons without clashes can be formulated as a graph coloring problem. Here each vertex represents a lesson, and an edge between two vertices represents that

the two corresponding lessons have at least one teacher or student in common. For any graph coloring problem we can find a corresponding timetabling problem. Hence the problem we consider is NP-hard.

2.1 The assets

We use the terms 'assets' for data describing the physical resources and time. Assets in the school timetabling problem are:

- *Time slots.* The time slots have a day attribute (Monday, Tuesday, etc.), and usually there are 5 to 10 time slots per day; this number can vary per day depending on the school and the educational system.
- *Students, Teachers*, and *Rooms.* Each of these is available only once per time slot.
- *Lessons.* These are the actual meetings between a group of students and one or more teachers. We assume that the students and teachers are preassigned. Throughout we assume that similar lessons, i.e. lessons to a group of students in the same subject, have to be scheduled on different days. A lesson has a duration, describing the number of consecutive time slots on the same day needed for this lesson. In practice the duration is one or two.
- *Room types.* Each room will have exactly one room type, and each lesson will have one room type. The meaning of this is that a lesson with room type A must be scheduled in a room of room type A.
- *Events.* Lessons that have to be scheduled at the same time slot, we call an *event.* We call the corresponding lessons *linked* lessons. This happens for instance if two groups are mixed because of level or sex. More intricate links exist, and are due to the requirement that certain optional lessons must be scheduled at the same time ([25]).
- *School class.* In most schools students are organized into base groups, which we call school classes; every student is in exactly one school class. Usually students in a school class take more or less the same lessons, while sometimes all students take the same lessons. In the last case we can speak of idle times of a school class, being equivalent to the idle times for any individual student in this school class.

2.2 The constraints

Below is a list of constraints that we encountered in practice. For each constraint we list whether it is considered as a hard constraint {H} or a soft constraint {S}, or can appear in both modes {HS}.

{H} (Events Constraint) Some lessons have to be taught simultaneously, leading to linked lessons (events).

{H} (Availability Constraint) A teacher or a room can be unavailable in one or more time slots.

{HS} (Off-Constraint) A teacher can have preferences for days off or off hours on specific days.

{S} (Day Load Constraint) A teacher or a student can prefer a minimum/maximum number of lessons per day.

{HS} (Day Idle Times Constraint) A teacher or a student can prefer a maximum number of idle times per day. Here an idle time is a time slot without a lesson for this teacher or student, but with a lesson earlier, and a lesson later on that day.

{HS} (Week Idle Times Constraint) A teacher or student can prefer a maximum number of idle times per week. For example we could ask for a teacher to have at most one idle time per day, but at most three idle times per week.

{HS} (NumberOfDaysConstraint) A teacher can prefer to have lessons on at most a maximum number of days, or at least a minimum number of days. For example, a part-time teacher can have the right to have at most three or four days with lessons.

3 Literature review

In most of the research in the area of high school timetabling, local search methods and meta-heuristics are used to find good solutions in reasonable computational time. These methods include simulated annealing [3], genetic algorithms [18,22,35], and tabu search [9,26,36,39]. The main characteristic of these algorithms is using simple moves/changes in defining the neighbourhood for their current timetable. Previous research on humanising automated scheduling systems [20,10,17,16] shows that sequential moves are used by expert timetablers to improve a timetable. This was the main motivation for our research.

Some researchers [30,11,12,37] have developed Integer Linear Programming (ILP) models for their high school timetabling problems. In [37], advanced integer programming techniques are used to solve relatively small class-teacher models (see [44]) to optimality. In [11,12], instances are studied more similar to our datasets. In [12], the authors introduce 'shifts' for teachers in a preprocessing phase. In this way the schedules for teachers are of good quality. Compared to the Dutch datasets we note the following differences:

- Students have highly individual schedules in the higher grades; hence the number of 'class-sections' is much higher in our datasets.
- Because of the many individual schedules, compact schedules are not feasible in the higher grades.
- In the Netherlands 80% of the teachers work part-time. We expect that the shift structure imposed has a bigger impact than in the Greek situation.

Summerizing we believe that solving ILP models for the datasets in Section 5 is still out of scope.

The concept of improvement graph, was introduced in Thompson and Orlin [41] and further examined by Thompson and Psaraftis [42]. Cyclic transfers are applied to a wide variety of combinatorial optimization problems including the traveling salesman problem [21,31,34], the quadratic assignment problem [8],

vehicle routing problems [4,23], the capacitated minimum spanning tree problem [6], the generalized assignment problem [48,49], parallel machine scheduling problems [5], and timetabling problems [27,1,2]. For a survey on very large neighbourhood search, cyclic transfers and their applications we refer the reader to [7,32]. Essential in all cases is that an admissible cycle in the improvement graph corresponds to a move to a neighbour of the current solution. The cost change of this move equals the cost of the cycle, which is the sum of the costs of the arcs in the cycle. For cycles that are *not* admissible this correspondence in cost is not guaranteed, or even worse: the corresponding move can be infeasible.

Unfortunately finding an admissible negative cost cycle in an improvement graph is an NP-hard problem, see Lemma 3.4.2 in [41]. Like Thompson and Orlin we believe that it still reasonable in practice to search for such cycles. In our opinion the cyclic transfer is very suitable as a search strategy transversal to local search methods like Simulated Annealing or Tabu Search. The cyclic transfer can search a large neighbourhood, and as such can correspond to an intensification step of the complete algorithm. On the other hand our experiments show the cyclic transfers will not generate good schedules starting from poor ones, see Section 6.

Cyclic transfers have been applied to timetabling problems only recently, see [27,1,2]. In the dissertation [27] the university course timetabling problem is discussed. The main objective here is to schedule the meetings of courses to time slots such that there are no clashes for teachers and students and the number of rooms used does not exceed the number of rooms available. A node in the improvement graph corresponds to a meeting of the course, and the resource attached to the node corresponds to the current time slot of the meeting. The technique used for finding the admissible negative cycles is dynamic programming.

In [1,2] the examination timetabling problem is studied. Here the main objective is to schedule exams to time slots to find a clsh-free timetable and such that the clashes of higher order between exams (proximity clashes) are minimized. Extra requirements can be obeying the room capacities and relations between different exams (precedence, distance in time). A node in the improvement graph corresponds to an exam, while the resource is the current time slot of the exam. A negative cycle is found with a modified label correcting algorithm as introduced in [6] starting from some initial nodes; this method assures that the selected negative cycle is admissible; however there is no guarantee that all admissible negative cycles are found.

4 The improvement graph

As stated above in the papers [27,1,2] the main objective is feasibility. For this reason the resource in the improvement graph can be a time slot: we only have to make sure that each time slot is scheduled well, i.e. we are looking for a clash-free schedule.

In high school timetabling we (of course) look for clash-free schedules as well. However this is not the end point. Clash-free schedules may abound, but can be unacceptable due to the poor quality. The quality of a schedule for a teacher or student is judged primarily on a day basis: each day the number and grouping of lessons should satisfy certain constraints, see Subsection 2.2. This difference in objective means that the resource 'time slot' is not suitable here. We describe the resources we use instead in the next subsection. In Subsection 4.2 we explain the construction of the improvement graph. Subsection 4.3 contains the basic lemma, that is the foundation of our algorithm. The algorithm itself is described in Subsection 4.4.

4.1 Cylic exchanges and the objective value

In our local search approach we use cyclic transfers as our moves. A cyclic exchange is a sequence of insertions and ejections. Here we investigate the influence of two consecutive shifts on the objective function. In the first shift, S_1, lesson L_1 is moved (from a time slot not on day D) to time slot t_1 on day D, while the next shift, S_2, removes lesson L_2 from time slot t_2 on day D (to another day). When studying the change in objective value, we merely have to study what happens to the cost functions of the teachers and students involved. So assume teacher T is involved in lesson L_1 or L_2; for students the situation is similar. If the cyclic transfer is executed, the schedule of teacher T changes. Our basic assumption is that the schedule of teacher T on day D is not effected by other shifts in the cyclic transfer. Our aim is to calculate the cost change for teacher T by looking at all pairs of consecutive shifts separately ('local changes'). We will demonstrate that this is not completely possible, and we describe how we treat such cases.

We distinguish three slightly different cases:

a. Teacher T is involved in both shifts, i.e. in the schedule of teacher T on day D lesson L_1 is added now at time slot t_1, while lesson L_2 disappears from time slot t_2. It is possible that $t_1 = t_2$. We need to recalculate the cost of the OffConstraints, DayIdleTimesConstraints for day D, and the WeekIdleTimesConstraints as far as valid for teacher T. The other types of constraints do not generate any change in cost. For the first two constraints it is sufficient to know the old and the new schedules of teacher T only on day D. For the WeekIdleTimesConstraints the situation is more subtle. The knowledge of the old and new schedules of day D suffice to find the *change* in the number of idle times throughout the week. However, if the cost is *not* proportional to the number of idle times, we cannot find the cost change, as the number of idle times might change on other days as well. For approximating the cost change due to the pair (S_1, S_2) we will assume that other parts of the cyclic transfer did not change the number of idle times. For example, if originally there are 3 idle times, and on day D we generate an extra idle time, we suppose that the number of idle times changes from 3 to 4. If the cost is quadratic in the number of idle times and the weight of

the soft constraint is 10, the cost change is estimated at $10 * (4^2 - 3^2) = 70$. If this happens on two days, we have assumed a cost change of 140 instead of the correct $10 * (5^2 - 3^2) = 160$.

b. Teacher T is not involved in lesson L_2. So teacher T will have an extra lesson on day D, namely L_1 at time slot t_1. Now we need to recalculate the cost of the OffConstraints, DayLoadConstraints, DayIdleTimesConstraints on day D as well as the WeekIdleTimesConstraints and the NumberOfDaysConstraints for teacher T. The cost change for the first three constraints can be calculated exactly. For the WeekIdleTimesConstraints the situation is similar to the first case. The NumberOfDaysConstraint only generates a non-zero cost change if lesson L_1 is the only lesson on day D. Moreover teacher T should either be at the *maximum* number of days with lessons for teacher T as specified by NumberOfDaysConstraint, or the teacher was below the minimum number of days with lessons. If a maximum is specified, we *always* assume that teacher T was already at this maximum, and penalize lesson L_1 accordingly. Note that this is not correct in all cases, especially for cases with only a few days of lessons for teacher T: in the cyclic exchange there could be several days which get a first lesson, all of them penalized separately instead of together. Regarding the minimum, we take no action at all, i.e. we neglect that one day extra might be planned for teacher T. The reason is that we want to avoid the double bonus, as the NumberOfDaysConstraint is usually highly weighted in the objective function.

c. Teacher T is not involved lesson L_1. So teacher T will have a lesson less on day D, namely L_2 disappears from time slot t_2. This situation is similar to the case above, with the NumberOfDaysConstraints treated in the same way with the minimum and maximum interchanged.

4.2 Construction of the improvement graph

In Subsection 2.2 we described the constraints that we consider. An important hard constraint is the EventsConstraint, which states that certain lessons are to be taught at the same time: such linked lessons we called events. Since we want to maintain feasibility at all times, we will move events instead of separate lessons. For this reason we continue with events.

Nodes and Arcs

The improvement graph will contain three types of nodes:

- A node for each event.
- A node for each time slot.
- One dummy node.

We add arcs according to the following scheme.

- We add an arc from the dummy node to each event node.
- We add an arc from each time slot node to the dummy node.

– We try to move each event node to a time slot ('A') on a different day. In trying to do this, three things can happen:
  - This can be done without moving another lesson, i.e. all involved students and teachers (and enough rooms) are available at time slot 'A'. In this case we add an arc from the corresponding event node to the time slot node of time slot 'A'.
  - This can be done by moving one (conflict) event to another day. In this case, we add an arc from the original event to the conflict event.
  - Several events have to be moved from time slot 'A'. In this case, no arc is added.

Note that due to the construction there are two types of cycles:

– Cycles that do not contain the dummy node. Hence they contain no TimeSlotNode as well, since a TimeSlotNode only has an arc to the dummy node. Consequently such cycle of length k has the form:

$$\text{Event}_1 \to \text{Event}_2 \to \cdots \to \text{Event}_k \to \text{Event}_1$$

Hence each event has a conflict event.
– Cycles that contain the dummy node. If the cycle has length k it is of the form:

$$\text{Dummy} \to \text{Event}_1 \to \text{Event}_2 \to \cdots \to \text{Event}_{k-2} \to \text{TimeSlot} \to \text{Dummy}$$

Hence Event_{k-2} is moved to the time slot corresponding to the TimeSlotNode, and has no conflict event.

Cost on arcs
The costs on the arcs are determined along the lines described in Subsection 4.1. We describe the interpretation of each arc. The cost on the arc corresponds to the cost change of the interpretation.

– An arc from the dummy node to an event node: the event (on day D) is removed. For the teachers and students in lesson L of the event, the schedule on day D changes, because lesson L moves to another day. The cost change is calculated as described in item c. of Subsection 4.1.
– An arc from the time slot node to the dummy node. This arc is added to turn a path ending in a time slot node to a cycle. Hence the cost is 0.
– An arc from an event node to a time slot node. The event is moved to time slot t, which corresponds to the time slot node, but no event on day D of time slot t is moved. For a teacher or student involved in lesson L of the event, in the schedule on day D the lesson L is added at time slot t. The cost change is calculated as in item b. of Subsection 4.1.
– An arc from event node to event node. Hence some linked lessons move to time slot t_1 on day D, while another group of linked lessons move from day D to another day. For the teachers and students involved in one of these lessons, we calculate the cost change according to Subsection 4.1.

Resources
The construction of the improvement graph shows that a cycle in this graph corresponds to a sequence of changes in the timetable. The change in the timetable we called the cyclic transfer. The cost of the cycle is the sum of the arc costs in the cycle. Of course, our aim is that the cycle cost reflects the cost change of the cyclic transfer. In Subsection 4.1 we explained that this works reasonably well if for each teacher and student the schedule on a certain day is changed only once. Hence we want the cycles we construct to satisfy this condition. We will force this property by adding *resources* to nodes and arcs, and we require that a cycle has disjoint resources. In the improvement graph a resource is a combination (teacher, day) or a combination (student, day). The way we add resources to nodes and arcs is as follows:

- For each event node (on day D) we consider the teachers and students of the linked lessons. Each one of these teachers and students is paired to D, and this pair is a resource which we add to the node.
- For each arc from an event node to time slot node t, we find all teachers and students in the lessons of the event, pair them to the day of the time slot node t and add these pairs to the arc.
- For each arc from an event node to another event node, we find all teachers and students in the lessons of the first event, pair it to the day of the second event and add these pairs to the arc.

4.3 Algorithm for finding negative cycles

Here we present a recursive method to determine a cycle of negative cost with disjoint resources. Our method is based on the following lemma.

Lemma.
Suppose $C = (N_0, N_2, N_{\ell-1})$ is a cycle with ℓ nodes and negative cost. Let c_i be the cost of the arc from node N_i to node N_{i+1}, where these indices are taken modulo ℓ.
Then there exists an index $i \in \{0, 1, 2, \ldots, \ell\}$ such that

$$\begin{aligned} c_i &< 0 \\ c_i + c_{i+1} &< 0 \\ c_i + c_{i+1} + c_{i+2} &< 0 \\ &\cdots \\ c_i + c_{i+1} + \cdots + c_{i+\ell-1} &< 0 \end{aligned} \tag{1}$$

Proof (by induction on ℓ).

If C is a negative cycle of length 2, then the result is clear, since $c_0 + c_1 < 0$ (C is a *negative* cycle), and hence either $c_0 < 0$ or $c_1 < 0$ (or both).

Suppose now that we have a negative cycle of length ℓ. If two consecutive arcs are both either non-negative or non-positive, we can merge them to a single arc

with cost the sum of the cost of the separate ones, and apply induction. So we can assume that the arcs in the cycle are alternating in sign (and consequently ℓ is even). We put indices such that the arc from N_0 to N_1 has negative cost. Now starting from index 0, consider the arcs two by two. These two arcs together have cost $c_{2i} + c_{2i+1}$, for $i = 0, 1, \ldots, \frac{\ell}{2}$. Clearly at least once this sum is negative, Since the total sum is negative, at least one of these pairs is negative as well. Merging these two arcs, we can apply induction to find the start index i such that all conditions in the equations (4.3) hold. Expanding the merged arcs again, we find our solution. □

For detecting a negative cycle, we use a recursive algorithm. We sort all the arcs in increasing order of their costs; each arc is tried as the starting arc for a cycle. In the construction of a cycle, we recusively add an extra arc to the directed path found so far. We avoid duplicate resources, and arcs that will lead to a directed path of non-negative cost. When the arc is added, we update the resources, and the total cost of the path. In the case that the newly added arc ends at the start of the first arc, we have an admissible negative cycle. If we look for the first descent, we return this cycle, otherwise we replace the best found if appropriate, and apply the 'partial gain' criterion: we do not allow any path to exceed the cost of the best found cycle. This might prevent us from obtaining the *steepest descent*, but the gain in computation time is too high to ignore. We call this descent 'partial gain descent'. If we get stuck (the path cannot be extended anymore), we backtrack. The correctness of this approach for finding a negative cycle is guaranteed by the previous lemma.

In practice we use one extra condition. We require the length of the cycle to be restricted by some number L; hence we allow only cycles with at most L arcs. In theoretical sense this is a major difference: without length restriction, the problem is NP-hard (see for example [13]), while with restriction the number of possible cycles is polynomial in the number of arcs. In practical sense, this restriction has a major effect on the computation time.

4.4 Algorithm and imperfections of the network model

The basic algorithm that we will run looks as follows:

```
function CyclicTransfers( TimeTable );
  Success := true;
  while Success do begin
    G := ImprovementGraph( TimeTable );
    Cycle := NegativeCycle( G );
    Success := Assigned( Cycle );
    if Success then begin
       TimeTable := ExecuteCycle( TimeTable );
       SaveToBestIfNeeded( TimeTable );
    end;
  end; % of while
```

```
  return BestTimeTable;
end.
```

Several remarks are in order here. The part within the while-loop we will call a *round*. We can continue the algorithm as long as we find a negative cycle, i.e we put no a priori limit on the number of rounds. The negative cycle is executed, no matter what the effect is on the cost of the schedule. Indeed, the cost of the new schedule can be higher than the original one, due to imperfections of the model, as explained in Subsection 4.1. This leads to the following two aspects.

- *Cycling.* To eliminate cycling we put the first two[1] events of the cycle at the beginning of a Tabu list. At the same time the events at the end of the list are removed as far as the length of the Tabulist exceeds maximum length, which we put at 15. As long as an event is in the Tabu list, it cannot be shifted (the corresponding event node is not created). If a new best timetable is found, we clear the Tabu list.
- *Hill climbing.* We investigate the algorithm as 'hill climber'. Especially when long cycles are allowed, after some time many negative cycles do not give rise to an improved timetable. If the algorithm is in hill climber setting, we stop the algorithm after five rounds without improving the best timetable.

5 Datasets

5.1 Real world data

To test the efficiency of our approach, we used 4 sets of real world data from three different schools (see Section 5). These datasets are available at `http://wwwhome.math.utwente.nl/~postgf/BenchmarkSchoolTimetabling/`. Problems K1 and K2 are defined in collaboration with the corresponding secondary school, where an expert timetabler builds schedules manually. The methods of this paper were not applied to them at that time, as these methods were developed afterwards. In 2007 parts of the schedule were generated using cyclic transfers. In 2008 the complete schedule was generated interactively using Tabu Search and cyclic transfers all the time; the improvements found by cyclic transfers usually were above 25%, and even up to 50%. In 2009 we used a hybrid search based on the Tabu search and the cyclic transfers, thus automating the interactive approach of 2008.

For the experiments that we describe here we constructed 10 "good" schedules, and 10 "bad" schedules using Tabu Search. On these initial schedules we perform the cyclic transfer experiments, controlling the maximum length of the cycles. In particular we perform hill-climbing experiments with 'first descent' and 'partial gain descent'. Finally we present some results based on a simple Tabu mechanism for escaping from local optima.

[1] We tried to include as few events as possible. When we added only the first event, cycling reappeared rather soon.

For our research we used four real datasets from schools; two schools called K and G from the Netherlands (school K with data K1 and K2 from two years), and one school E from the UK. The instances have the following basic characteristics. Here the students are only the students with optional subjects. Precise data for the other students is not provided by the school.

Dataset	K1	K2	G	E
Days	5	5	5	5
Time slots	38	37	44	30
Teachers	86	87	136	76
Classes	36	38	84	67
Students	453	498	846	0
Room types	2	2	1	1
Lessons	1178	1238	2800	1227

Table 1. Data of the instances K1, K2, G and E.

The objective function was set by us, based on communications with the planners, along the lines described in Subsection 2.2. In all datasets we try to minimize the number of idle times of teachers and school classes, whilst trying to maintain a balanced spreading of the lessons. There are two important aspects in the Dutch situation:

- The students in higher grades have optional subjects. The number of optional subjects, and the number of lessons can vary between students, even within the same grade and level. As a consequence of this, it is impossible to create schedules for (all) these students without idle times. In contrast we require compact schedules for the school classes in lower grades. The usual way of handling the optional subjects is to link their lessons to events. The linked lessons have to be such that the students in these lessons are disjoint.
- The majority of teachers work part-time. If a teacher has to teach x lessons, we require that these lessons take place on $\frac{1}{5}x$ (rounded up) days, but not more than 5.[1]. These teachers can have preferences for which days off as well. In addition we try to spread these lessons evenly over the teaching days.

In the Netherlands the teachers usually are preassigned to the lessons. In the English situation, this is not the case. The dataset E that we use, is in fact the end result of the scheduling process, with the time slots removed (rooms play no role). We describe some further details for each of the cases.

[1] Full-time teachers have lessons for 26 time slots. This rule will give them five days with lessons, as required. A part-timer with lessons for 18 time slots will be entitled to have one day off.

A possible scaling for weights could be

- weight is 0: the constraint is ignored;
- weight is 1: it would be nice to respect this constraint;
- weight is 10: the constraint is important;
- weight is 100: the constraint is very important;
- weight is 1000: the constraint is essential (almost a hard constraint).

This scaling is used in the datasets presented in Section 5.

K1
The dataset K1 is from the year 2003/2004. It represents an average Dutch secondary school, with around 1000 students, in 36 school classes. Of these students, 453 are in the higher grades with optional subjects. In this dataset, and in K2, there is one special room type for gym lessons; there are two rooms, which are in use for 100 % of the time. There are eight time slots per day, but on Thursday there are only six time slots available, due to the weekly staff meeting. The last time slot on Friday should be avoided, but is not forbidden (in K2 it is). The dataset contains 928 events.

K2
This dataset is from 2005/2006, from the same school as K1. Between 2003 and 2005 the school organization changed considerably. The classes in the lower levels have two by two combined lessons, giving rise to two linked lessons. For this reason there are now only 762 events to schedule, but now of higher complexity.

G
This is a dataset of a large school, with approximately 2000 students, of which 846 have optional subjects. Room requirements are not provided. Special attention has to be paid to scheduling the school classes on all weekdays. Originally there were 10 time slots (of 40 minutes) per day. Since the school classes have around 35 lessons, it is possible to have a day without lessons. This is not acceptable. Similarly the last time slot, and time slot 9 on Friday should be avoided. In the dataset, we removed these time slots, leaving 44 in total. It turns out that it is easy to find feasible schedules within these 44 time slots. There are 2186 events, of which 38 are fixed.

E
This dataset from an English school is very tight. All lessons should be scheduled in five time slots per day, leading to compact schedules for students and most teachers. As described above we preassigned the teachers to the lessons, which is normally done afterwards. No feasible solutions were found by our algorithms. For this reason we added an extra time slot to each day, and penalize its use by a penalty of 1000. Even in the best known solution, the sixth time slot is used 17 times. The number of events is 543.

6 Experiments and results

In the previous sections we described the problem we study and the algorithm to improve previously found schedules. In this section our purpose is to show that the method we propose can improve the schedules considerably. These improvements are shown to be relatively stable in different datasets. In Section 6.1 we describe the basic schedules on which all tests were performed. In Section 6.2 we give the results for cyclic transfers algorithm in 'hill climbing' mode. In Section 6.3 we turn to some further experiments, where more room is given to diversifications.

6.1 Experimental set-up

To perform our experiments for the four datasets E, G, K1, and K2, we constructed initial schedules. In these timetables all lessons are scheduled to time slots, in such a way that rooms are available for all lessons. The way we generate these solutions is by a Tabu search algorithm, which we will not describe here. The best solutions produced by this Tabu search algorithm are of good quality, comparable to or better than hand-made schedules, at least for the datasets K1 and K2. A weakness of the algorithm is its greediness: the algorithm places lessons one by one, not being aware of global aspects of the schedule. Due to this one can expect that cyclic transfers are able to improve the schedules found.

For all four datasets, we performed several runs of the constructive algorithm. From the constructed schedules we select the 10 best results ("good schedules"), as well as a sequence of 10 results ("bad schedules") of which the cost is approximately twice that of the 10 best schedules. Our starting point is summarized in Table 2. For each dataset we list in the columns:

Set	Good schedules	Bad schedules	Best known	Nodes	Arcs
K1	1665 - 1933 (1822)	2600 - 2892 (2745)	1287	924	10280
K2	1308 - 1579 (1435)	2875 - 3659 (3374)	1082	786	5542
E	24488 - 29642 (27855)	43166 - 48416 (45596)	19760	574	3169
G	630 - 680 (652)	1196 - 1551 (1380)	468	2270	31694

Table 2. Start schedules and size of network per instance.

- the range of the good schedules: best - worst (average),
- the range of the bad schedules: best - worst (average),
- the overall best schedule known to us,
- the number of nodes in the improvement graph,
- the (approximate) number of arcs in the improvement graph.

The best known solutions were found in an ad hoc way: running a mix of algorithms with different parameters, keeping the best ones, and trying to improve those.

The number of nodes in the improvement graph is less than the number of lessons listed in Table 1, although the nodes include time slot nodes as well. This is partially due to skipping lessons that are fixed, or occupy two time slots. More important is that in the network we use events, instead of lessons. The number of arcs varies a little bit with the current schedule, since the admissible shifts depend on the number of conflict lessons, see Subsection 4.2. The amounts presented below correspond to the start of the run of the first (best) good schedule.

The datasets differ a lot in size: the smallest set (E) contains only 25 % of the nodes and 10 % of the arcs of the biggest set (G). We will see in Table 5 that this has a major effect on the running times. The experiments were performed on a Dell Latitude D810 with a 2.13 GHz processor and 2 GB of RAM, working under Windows XP. The implementation was done in Delphi 7.

6.2 Experiments with cyclic transfers as hill climber

The first experiments we discuss are the basic cases of a fixed maximum cycle length L, and stopping the run after five rounds without improvement. As in Subsection 4.4, we call this cyclic transfer as 'hill climber'. We apply the algorithm to the 10 good and to the 10 bad schedules for L ranging from 2 to 12. Only the dataset G for partial gain descent and $L = 12$ and bad schedules were not completed; the third instance was stopped after 12 hours of running time.

The results for the good schedules are displayed in Table 3 for partial gain and first descent, respectively. The tables display the average cost per dataset, as a percentage of the average initial cost.

L	start	2	3	4	5	6	7	8	9	10	11	12
K1	100	96.4	95.1	94.8	94.9	92.9	92.1	93.1	92.2	91.9	91.8	91.7
	100	96.5	95.2	95.5	96.1	93.4	93.7	94.7	92.8	95.2	94.0	94.7
K2	100	95.6	94.8	94.3	93.8	93.6	93.5	93.9	93.6	93.7	93.6	93.6
	100	96.2	94.7	94.6	94.1	93.8	93.9	93.4	93.2	93.6	93.9	93.6
E	100	97.4	97.3	95.1	94.6	93.3	93.2	92.6	92.5	92.1	91.6	91.2
	100	97.5	97.0	94.6	94.5	92.4	93.1	92.7	92.8	92.6	92.9	93.3
G	100	94.2	91.7	91.3	89.4	89.1	89.7	90.5	87.1	88.8	88.0	89.3
	100	94.3	92.3	93.5	91.8	94.3	93.9	95.8	96.4	97.4	97.8	97.3

Table 3. Results for the good schedules and both descents (hill climbing).

For example in Table 3 we have for K1 the numbers 100, 96.4, 95.1, etc. in the second row. The row gives the results for partial gain descent, while the next row is for first descent. The number 100 represents that originally the average cost is

100 % of the average cost 1822 (see Table 2), while the number 96.4 represents that after applying the algorithm for $L = 2$, the average cost is 96.4 % of 1822 (so 1756), and with $L = 3$ the average cost is 95.1 % of 1822, hence 1733. We display only the averages, as this seems enough to analyse the performances of the algorithms; the results on the individual schedules are homogeneous in the sense that the standard deviations of these remain in the same order of magnitude.

A similar comparison for the bad schedules can be found in Table 4.

L	start	2	3	4	5	6	7	8	9	10	11	12
K1	100	95.3	93.6	93.3	93.1	92.2	92.1	91.9	90.7	90.8	90.5	90.8
	100	95.5	93.6	93.0	93.0	92.1	91.8	93.3	92.8	94.4	94.8	93.2
K2	100	85.7	79.6	77.4	76.1	75.2	76.3	75.4	74.4	74.0	73.3	73.7
	100	86.1	79.3	77.0	77.8	76.6	79.9	82.2	78.4	79.9	80.9	79.4
E	100	96.8	96.1	95.6	95.0	93.6	94.0	92.7	92.5	92.5	92.5	92.8
	100	96.8	95.9	95.1	94.9	94.2	93.6	92.2	92.5	92.5	92.7	92.5
G	100	92.6	89.4	83.7	77.5	74.4	72.2	74.3	73.2	74.9	74.2	—
	100	92.4	88.9	86.0	84.1	81.6	85.8	88.4	91.5	95.6	93.1	96.6

Table 4. Results for the bad schedules and both descents (hill climbing).

The first observation is that the largest L does not always give the best results; this is only the case for 2 of the 8 tests, both for partial gain descent. For first descent we have the best results four times for $L = 6$, and once for $L = 5$, $L = 7$, $L = 8$, and $L = 9$. For all tests with first descent, the best average of the cases $L = 10, 11, 12$ is worse than the best average for $L = 6, 7, 8$. This behaviour seems to be due to the hill climbing aspect of our cyclic transfers. Looking at dataset G, where the differences are the largest, we see that the number of executed cycles for the bad schedules drops from an average of 53 for $L = 6$ to 13 for $L = 12$. Hence we conclude that the limitations of our model take their toll here; the algorithm, being in hill climbing mode, was stopped, because the last five cycles executed did not yield a better result. As explained in Subsection 4.4, the imperfections of the improvement graph are due to combined effects of different arcs in a cycle. If cycles are long, the chance of interference becomes larger.

A second observation is that partial gain descent gives better results than first descent. Of the 87 comparisons that can be made, partial descent performs better with 0.5% or more in 45 cases, while for first descent this is only 6 times the case. Even if we limit the range to $L \leq 8$, we see a score of 22 for partial gain versus 6 for first descent, out of 56 cases to compare. Note that for $L = 2$ and $L = 3$ the results do not differ very much. For $L = 2$ partial gain descent performs 6 times (out of 8) better than first descent, while for $L = 3$ first descent is better in 5 cases. In Table 5 we compare the running times between partial gain descent and first descent. The times listed are the average running times

per schedule in seconds. As expected the first descent is faster, but remarkably not very much for the datasets K1, K2, and E. For dataset G the difference is growing with L: for small $L = 2$ the times are more or less the same, while for $L = 12$, the factor is $1608/275 \approx 5.8$. This is partly due to the number of rounds the algorithm executes, but mostly to the time per round: for $L = 2$ the running time per round is 1.75 seconds for partial gain descent, and 1.39 seconds for first descent, while for $L = 12$ the running times per round are 59.6 and 17.1 seconds, respectively.

We omitted the tables with the running times for the bad schedules. These running times are higher than those for the good schedules, because of the increase in the number of rounds. This happens especially for K3 (increase of 250%) and G (increase of 400%), the cases where even the relative improvements are much higher, compare Table 3 and Table 4.

L	2	3	4	5	6	7	8	9	10	11	12
K1	4.3	5.0	5.6	5.4	5.6	6.2	6.3	6.3	6.5	7.5	16.9
	3.8	4.5	4.5	4.3	6.2	6.6	6.4	7.1	4.5	7.2	6.0
K2	3.4	3.9	4.3	4.0	4.2	4.3	4.1	4.2	4.6	5.2	6.9
	2.5	3.4	3.6	3.7	4.0	4.0	4.7	5.0	5.0	5.1	6.0
E	1.2	1.6	1.6	1.6	1.8	1.8	1.8	2.0	2.0	2.0	2.4
	1.3	1.5	1.7	1.7	1.9	1.8	1.8	2.0	2.0	2.1	2.0
G	47	60	53	63	54	62	53	118	249	591	1608
	43	60	38	56	39	36	35	43	53	88	275

Table 5. Running times for the good schedules and both descents (hill climbing).

6.3 Further experiments with cyclic transfers

In Subsection 6.2 we broke off the computation of the cyclic transfers after 5 rounds without improvement. This was done to make the cyclic transfe rs algorithm work (more or less) as a hill climber. Due to our choice to execute any cycle that has negative cost in the improvement graph, without checking the effect on the timetable in advance, some changes are not climbing the hill. If several of these moves, say five or more, are executed, this can be viewed as a kind of diversification. Hence, it is reasonable to expect that we will obtain better results in this case, clearly at the cost of larger running times. In fact, it is not even clear that the algorithm will stop. Indeed, while for the datasets K1, K2, and E this poses no problem up to $L = 12$, for G the running times increase rapidly. For this reason, we omitted the cases $L > 10$ for dataset G. We combined the results for partial gain and first descent in Table 6.

We observe that the differences between partial gain descent and first descent diminished. This is mainly due to the fact that the results for first descent improved quite a lot; only in 15 of the 42 cases, of which 7 for dataset E, there

Set	start	2	3	4	5	6	7	8	9	10	11	12
K1	100	96.4	95.1	94.8	94.9	92.9	92.1	92.1	91.4	91.9	91.1	91.7
	100	96.5	95.2	95.0	95.5	93.0	93.6	93.1	91.0	93.8	92.3	92.7
K2	100	95.6	94.8	94.3	93.8	93.6	93.5	93.2	92.9	93.2	93.2	93.2
	100	96.2	94.7	94.2	94.1	93.8	93.7	93.4	93.0	93.1	92.6	92.8
E	100	97.4	97.3	95.1	94.6	93.3	93.2	92.6	92.5	92.1	91.6	91.2
	100	97.5	96.9	94.6	94.5	92.4	93.1	92.7	92.8	92.5	92.8	92.8
G	100	94.2	91.7	90.8	89.2	87.5	86.1	85.2	83.8	82.5	—	—
	100	94.3	92.0	90.9	89.3	87.7	86.7	85.0	83.9	83.2	—	—

Table 6. Results for the good schedules and both descents.

is no improvement. For partial gain descent, the improvements are much less. For dataset E there is no improvement at all, while for the K datasets, there are only small improvements for $L \geq 8$. Only for dataset G, the improvements are substantial.

In Table 7 we present the average running times. This table should be compared to Table 5, where the same is listed for the hill climber. For small L the running times did not increase very much[1], but for larger L the differences grow, especially for first descent. This is in agreement with the fact that for small L the computation is stopped, because no new cycle was found, while for larger L, it is more often the case that cycles are still found, but they do not improve the result. For large L the results of partial gain descent and first descent are comparable. Note however that the running times in first descent are usually larger than in the partial gain descent.

L	2	3	4	5	6	7	8	9	10	11	12
K1	4.3	5.0	5.6	5.4	5.6	6.2	6.7	7.1	6.5	8.5	18.4
	3.8	4.6	4.8	5.6	6.8	6.8	9.4	11.0	9.4	20.0	20.3
K2	3.4	4.0	4.3	4.0	4.2	4.3	4.7	5.0	5.4	5.6	7.4
	2.5	3.5	4.1	3.7	4.1	4.1	5.1	5.3	6.0	6.4	7.1
E	1.2	1.6	1.6	1.6	1.8	1.9	1.9	2.0	2.0	2.0	2.5
	1.3	1.5	1.7	1.7	1.9	1.8	1.8	2.0	2.2	2.3	2.5
G	47	59	62	74	73	105	139	240	519	–	–
	43	63	63	84	105	192	244	596	5007	–	–

Table 7. Running times for the good schedules and both descents.

[1] Note that for dataset G and $L = 3$ the hill climber seemed to take more time; this is due to rounding.

6.4 Using oscillating descent

The results in Table 6 show that it is useful to continue the search based on our neighbourhood, even if for some time no better solution is found. This situation seems a paradox: we execute a negative cycle, but initially the obtained solutions are worse. Still, after doing this for some time, we end up with better solutions. In other words, we try to do local optimization, but end up doing diversification, leading to improvements later on. In this subsection we try to improve this behaviour a little bit by allowing small cost increases from time to time. It is not meant as a systematic study, but arose out of curiosity: is it possible to improve the results even further, using the same neigbourhoods?

The method we employ is the following: we start performing our basic algorithm with partial gain descent. If no better timetable is found for some time, we switch to first descent, and allow a (small) cost increase (of 10 maximally) per cycle. We chose an 'oscillation period' of 15 rounds, of which we wait 11 rounds before switching to first descent mode. However:

- Each time the new timetable has the same cost as the previous timetable, we wait one round less.
- Each time the new timetable is better than the previous timetable, we wait one round more.

After four rounds of first descent, we turn back to partial gain descent, and start counting again. The period of four rounds with first descent are meant to improve the diversification. Switching from partial gain descent to first descent has two advantages: first descent is faster, and first descent might choose different types of cycles to execute. The parameters 11 (for partial gain mode) and 4 (for first descent mode) are rather arbitrary, and were set with the idea that we shouldn't wait too long, and shouldn't divert too much. In pseudo code our method looks as follows.

```
function OscillatingDescent( TimeTable );
begin
  while time left do begin
    i := 11;
    while i > 0 do begin
      NewTimeTable = CyclicTransfersPartialGain( TimeTable );
      if NewTimeTable.Cost > TimeTable.Cost then i := i - 1;
      if NewTimeTable.Cost = TimeTable.Cost then i := i - 2;
      TimeTable := NewTimeTable;
    end;
    for i := 1 to 4 do
      TimeTable := CyclicTransfersFirstDescentMaxCost10( TimeTable );
  end; % of time left
  return BestTimeTable;
end;
```

The results are shown for the good schedules in Table 8 and for the bad schedules in Table 9. We fixed a running time of 100 seconds for the datasets K1, K2 and G, and 250 seconds for dataset G.

Set	start	2	3	4	5	6	7	8	9	10	11	12
K1	100	94.4	92.5	91.7	89.8	89.1	86.7	87.7	87.6	86.7	85.9	88.0
K2	100	94.7	91.8	89.3	89.9	89.0	87.7	89.3	88.1	89.2	89.3	89.4
E	100	96.8	94.8	90.7	90.0	88.4	89.5	89.1	87.2	88.2	88.4	87.8
G	100	92.8	88.5	87.5	85.5	83.8	82.9	83.6	84.3	88.7	90.7	94.4

Table 8. Results for the good schedules and oscillating descent.

Comparing to the results in the Tables 4 and 6 we see a significant improvement of around 5% for 7 out of the 8 cases. Only for the good schedules of dataset G the difference is smaller: the best result for partial gain descent is 82.5%, while in oscillating descent it is 82.9%. Note however that the time needed to reach 82.5 % was 519 seconds, which is twice as much as in oscillating descent.

Another result which is prominent is that the best results are reached for $L = 7$ in 5 of the 8 cases. Hence it pays off to use cycles longer than 3 or 4, but in this phase of the optimization it seems useless to use cycles of length more than 10.

Set	start	2	3	4	5	6	7	8	9	10	11	12
K1	100	94.1	90.4	88.1	88.6	86.2	85.6	86.2	84.1	84.4	86.1	86.2
K2	100	83.1	74.1	71.4	71.5	69.7	68.3	68.6	69.1	69.8	70.5	72.8
E	100	95.1	93.5	90.1	89.0	88.5	88.3	87.7	88.9	88.7	89.1	88.4
G	100	89.8	85.3	79.1	73.8	70.8	68.7	71.8	75.7	80.7	85.7	91.2

Table 9. Results for the bad schedules and oscillating descent.

7 Conclusions

Our aim was to show how cyclic transfers can be used in high school timetabling. Though the improvement graph cannot exactly model our problem, we are able to use it in a profitable way. Especially for long cycles, we encounter serious problems which usually lead to the cost changes being underestimated.

The algorithm we use for finding the negative cycles in the improvement graph, is a recursive method based on the lemma in Subsection 4.3. For the smaller graphs of the datasets K1, K2, and E the performance is above expectation. For the larger dataset G the performance becomes a problem if long cycles

are allowed. An obvious way for performance improvement is updating the network after a cycle is executed, instead of rebuilding as we do now. Especially when using first descent, generating the network, and more specifically calculating the costs of the thousands of arcs will take time, which can be avoided by more than 90% in most cases.

For the cases K1, K2, and E the oscillating descent gives improvements of more than 10% for maximum cycle length 7. For the bad schedules of the datasets K2 and G the improvements are over 30%. For dataset G the higher improvements can be explained from the size of the dataset, and the way the original schedules were created: for all datasets we use a similar generation time, which, apparently, led to relatively poor solutions for G. For K2, we can note that the bad schedules compared to the good ones, are of poorer quality than for other datasets.

We did not attempt fine tuning of our methods, as far as parameter settings are concerned. We are convinced that adapting parameters to the datasets will give even better results. We believe that the way it is presented now is clean and convincing.

Acknowledgement
We thank Dr Maria Kavanagh for proof reading of the manuscript.

References

1. S. Abdullah, S. Ahmadi, E.K. Burke, and M. Dror, 'Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling', OR Spectrum **29**, pp. 351–372, (2007).
2. S. Abdullah, S. Ahmadi, E.K. Burke, M. Dror, B. McCollum, 'A tabu-based large neighbourhood search methodology for the capacitated examination timetabling problem', Journal of the Operational Research Society **58**, pp. 1494–1502, (2007).
3. D. Abramson, 'Constructing school timetables using simulated annealing: sequential and parallel algorithms', Management Science Vol. **37**, pp. 98–113, (1991).
4. R. Agarwal, R. Ahuja, G. Laporte, and Z. Shen, 'A composite very large-scale neighborhood search algorithm for the vehicle routing problem', in Handbook of Scheduling: Algorithms, Models and Performance Analysis, chapter 49. Chapman & Hall/CRC, Boca Raton, FL, (2003).
5. R. Agarwal, O. Ergun, J. Orlin, and C. Potts, 'Solving parallel machine scheduling problems with variable depth local search', Working Paper, Operations Research Center, MIT, Cambridge, MA, (2004).
6. R. Ahuja, J. Orlin, and D. Sharma, 'Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem', Mathematical Programming **91**, pp. 71-97, (2001).
7. R.K. Ajuha, O. Ergun, J.B. Orlin, and A.P. Punnen, 'A survey of very large-scale neighborhood search techniques', Discrete Applied Mathematics **123**, pp. 75–102, (2002).
8. R. Ahuja, K. Jha, J. Orlin, and D. Sharma, 'Very large-scale neighborhood search for the quadratic assignment problem', Working Paper, Operations Research Center, MIT, Cambridge, MA, (2002).

9. R. Alvarez-Valdes, G. Martin, and J.M. Tamarit, 'Constructing good solutions for the Spanish school timetabling problem', Journal of Operational Research Society **47**, pp. 1203–1215, (1996).
10. S. Ahmadi, R. Barone, E. Burke, P. Cheng, P. Cowling, and B. McCollum, 'Integrating human abilities and automated systems for timetabling: A competition using STARK and HuSSH representations at the PATAT 2002 conference', Proceedings of the 4th international conference on the practice and theory of automated timetabling (PATAT 2002), KaHo St.-Lieven, Gent, pp. 265–273, (2002).
11. T. Birbis, S. Daskalali, and E. Housos, 'Timetabling for Greek high schools', Journal of the Operational Research Society **48**, pp. 1191–1200, (1997).
12. T. Birbis, S. Daskalali, and E. Housos, 'School timetabling for quality student and teacher schedules', Journal of Scheduling, DOI 10.1007/s10951-008-0088-2, (2008).
13. N. Boland, J. Dethridge, and I. Dumitrescu, 'Accelerated label setting algorithms for the elementary resource constrained shortest path problem', Operations Research Letters **34**, pp. 58–68, (2006).
14. E.K. Burke and S. Petrovic, 'Recent research directions in automated timetabling', European Journal of Operational Research **140**, pp. 266–280, (2002).
15. M.W. Carter and G. Laporte, 'Recent developments in practical course timetabling', in 'Practice and Theory of Automated Timetabling II', Lecture Notes in Computer Science **1408**, Springer Verlag, E. Burke and M. Carter (Eds.), pp. 3–19, (1998).
16. P. Cheng, R. Barone, P. Cowling, and S. Ahmadi, 'Opening the information bottleneck in complex scheduling problems with a novel representation: STARK diagrams', Diagrammatic representations and inference: Second International Conference, Diagrams 2002, pp. 264–278, (2002).
17. P. Cheng, R. Barone, S. Ahmadi, S., and P. Cowling, 'Integrating human abilities with the power of automated scheduling systems: Representational epistemological interface design', AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments, (2003).
18. A. Colorni, M. Dorigo, and V. Maniezzo, 'Metaheuristics for high school timetabling', Computational Optimization and Applications **9**, pp. 275–298, (1998).
19. T.B. Cooper and J. Kingston, 'The solution of real instances of the timetabling problem', The Computer Journal **36**, pp. 645–653, (1993).
20. P. Cowling, S. Ahmadi, P. Cheng, and R. Barone, 'Combining Human and Machine Intelligence to Produce Effective Examination Timetables', Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL2002), pp 662–666, (2002).
21. V. Deineko and G. Woeginger, 'A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem', Mathematical Programming **87**, pp. 255–279, (2000).
22. A. Drexl and F. Salewski, 'Distribution requirements and compactness constraints in school timetabling', European Journal of Operational Research **102**, pp. 193–214, (1997).
23. O. Ergun, 'New neighborhood search algorithms based on exponentially large neighborhoods', PhD dissertation, Massachusetts Institute of Technology, Cambridge, MA, (2001).
24. O.B. de Gans, 'A computer timetabling system for secondary schools in the Netherlands', European Journal of Operational Research **7**, pp. 175–182, (1981).
25. P. de Haan, R. Landman, G. Post, and H. Ruizenaar, 'A case study for timetabling in a Dutch secondary school', in 'Practice and Theory of Automated Timetabling

VI', E. Burke and H. Rudová (Eds.), Lecture Notes in Computer Science **3867**, Springer Verlag, pp. 267–279, (2007).
26. A. Hertz, 'Tabu search for large scale timetabling problems', European Journal of Operational Research **54**, pp. 39–47, (1991).
27. K. Jha, 'Very Large-scale Neighborhood Search heuristics for combination optimization problems', Ph.D. dissertation, University of Florida (2004).
28. J. H. Kingston. 'Modelling timetabling problems with STTL', in 'Practice and Theory of Automated Timetabling III', E.K. Burke and W. Erben (Eds), Lecture Notes in Computer Science **2079**, Springer-Verlag, pp. 309–321, (2001).
29. J.H. Kingston, 'A tiling algorithm for high school timetabling', in 'Practice and Theory of Automated Timetabling V', E. Burke and M. Trick (Eds.), Lecture Notes in Computer Science **3616**, Springer Verlag, pp. 208–225, (2005).
30. N.H. Lawrie, 'An integer linear programming model of a school timetabling problem', The Computer Journal **12**, pp. 307-316, (1969).
31. S. Lin and B. Kernighan, 'An effective heuristic algorithm for the traveling salesman problem', Operations Research **21**, pp. 498516, (1973).
32. C. Meyers, J.B. Orlin, 'Very large-scale neighborhood search techniques in timetabling problems', in: 'Practice and Theory of Automated Timetabling VI', E. Burke and H. Rudová (Eds.), Lecture Notes in Computer Science **3867**, pp. 24–39, (2007).
33. G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngas, C. Nurmi, D. Ranson and H. Ruizenaar. "An XML format for Benchmarks in high School Timetabling", in 'Proceeding of the 7th international conference on the Practice and Theory of Automated Timetabling (PATAT 2008)'.
34. A. Punnen and S. Kabadi, 'Domination analysis of some heuristics for the traveling salesman problem', Discrete Applied Mathematics **119**, pp. 117-128, (2002).
35. G. Ribeiro Filho and L. A. Nogueira Lorena, 'A constructive approach to school timetabling', in 'EvoWorkshop 2001', E.J.W Boers et al. (Eds), Lecture Notes in Computer ScienLawr69ce **2037**, Springer Verlag, pp. 130–139, (2001).
36. H.G. Santos, L.S. Ochi, and M.J.F. Souza, 'An efficient tabu search heuristic for the school timetabling problem', in 'WEA 2004', C.C. Ribeiro and S.L. Martins (Eds.), Lecture Notes in Computer Science **3059**, Springer Verlag, pp. 468-481, (2004).
37. H.G. Santos, E. Uchoa, L.S. Ochi, and N. Maculan, 'Strong Bounds with Cut and Column Generation for Class-Teacher Timetabling', Proceedings of The 7th International Conference on the Practice and Theory of Automated Timetabling PATAT 2008, (2008).
38. A. Schaerf, 'A survey of automated timetabling', Artificial Intelligence Review **13**, pp. 87–127, (1999).
39. A. Schaerf, 'Local search techniques for large high school timetabling problems', IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **29**, pp. 368–377, (1999).
40. K.A. Smith, D. Abramson, and D. Duke, 'Hopfield neural networks for timetabling: formulations, methods and comparative results', Computers & industrial engineering **44**, pp. 283–305, (2003).
41. P.M. Thompson and J.B. Orlin, 'The theory of cyclic transfer', Working paper OR200-89, Operation Research Center, MIT, Cambridge, MA, (1989).
42. P.M. Thompson and H.N. Psaraftis, 'Cyclic transfer algorithm for multivehicle routing and scheduling problems', Operations Research **41**, pp. 935–946, (1993).
43. C. Valouxis and E. Housos, 'Constraint programming approach for school timetabling', Computers & Operations Research **30**, pp. 1555–1572, (2003).

44. D. de Werra, 'An introduction to timetabling', European Journal of Operational Research **19**, pp. 151–162, (1985).
45. D. de Werra, 'On a multiconstrained model for chromatic scheduling', Discrete Applied Mathematics **94**, (1999).
46. R.J. Willemen, 'School timetable construction; algorithms and complexity', PhD-thesis, Technical University Eindhoven, The Netherlands, (2002).
47. M. Wright, 'School timetabling using heuristic search', Journal of Operational Research Society **47**, pp. 347–357, (1996).
48. M. Yagiura and T. Ibaraki, 'Recent metaheuristic algorithms for the generalized assignment problem', in Proceedings of the Twelfth International Conference on Informatics Research for Development of Knowledge Society Infrastructure, pp. 229–237, Kyoto, Japan, (2004).
49. M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover, 'A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem', Discrete Optimization **1**, pp 87–98, (2004).