A Strictly Combinatorial Approach to the University Exam Scheduling Problem

Eddie Cheng, Raymond Kleinberg, Serge Kruk,
William Lindsey, and Daniel Steffy

May 5, 2004

Keywords: Scheduling, NP-Hard, Combinatorial Optimization

Abstract

This paper gives a combinatorial approach to solving the student exam scheduling problem. The problem is to generate schedules that satisfy hard constraints while minimizing soft constraints. This problem is NP-Hard. The problem is decomposed into stages that include finding stable sets, weighted bipartite matchings, max-flow min-cut, and pathfinding in hyper graphs. We describe our method and discuss our results and implementation.

1 Background

The scheduling of exams is a common problem faced by educational institutions. Even in the simplest forms of this problem, determining if a feasible schedule exists is NP-Complete. Many others have previously studied this problem, and some recent papers include [2, 3, 6, 7, 8, 9, 12, 13, 14, 15, 17]. Some survey papers are [1, 5, 4]. Many previously attempted methods include Genetic Algorithms [1, 4, 7, 13, 14, 15], various hill climbing and local search strategies[8, 9, 17], constraint based approaches [6], and some that include combinations of these [3]. In this paper we will give a strictly combinatorial approach to solve a specific exam scheduling problem. The specifics of the problem we solve are based on an international timetabling competition held by Practice and Theory of Automated Timetabling (PATAT) [http://www.asap.cs.nott.ac.uk/ASAP/ttg/patat-index.html] in 2002/2003.

2 Problem

The student exam scheduling problem is a standard event scheduling problem. Given sets of exams and students, each student has a schedule of exams they must attend. Furthermore, these exams must be scheduled into a set of timeslots such that no student has multiple simultaneous exams. We will look at solving a more specific and complicated version of this problem, as proposed by PATAT. We are given:

- A Set of Exams
- A Set of Students
- A Set of Rooms
- A Number of Timeslots

With each of these sets comes additional information interrelating them. Each student has a schedule of exams they must attend. Each room has multiple features, including room size, and several additional features such as presence of a digital projector or dry erase board. Each exam has a set of required features including the number of students attending it, along with requirements regarding the presence of other features such as a projector. In this arrangement, rooms can only accommodate exams if they have enough seats to fit all the attending students, and have at least all the features required by the exam. For our problem we will consider timeslots for one week, arranged nine per day for five days. A feasible schedule would be the assignment of events to rooms within the limited number of timeslots such that: no students have to write multiple simultaneous exams and every room can accommodate each exam assigned to it. We will call the constraints required for a feasible schedule hard constraints.

In addition to finding a feasible schedule, there are other factors we might find desirable when finding a schedule. For example, if we can find multiple feasible schedules, we may want to decide when one is *better* than another. There are many factors one could consider when evaluating which schedule is the best. For our specific problem we look to a set of soft constraints that we will try to avoid violating, but for which violation of them will not result in infeasibility. From these soft constraints we can derive a cost function that computes a penalty for any feasible schedule based on the number of soft constraints it violates. In our case the soft constraints are as follows:

- Avoid students having a block with three consecutive exams
- Avoid students having a single exam in one day

- Avoid students having an exam in the last timeslot of the day

Each occurrence of any of these situations is considered equally bad, so the value of the cost function associated with a feasible schedule represents the number of times each soft constraint is violated. For example a feasible schedule with a cost function value of zero would be an optimal solution for any problem.

The specific instances we set out to solve were of medium size. They were large enough that brute force style methods would be utterly useless, yet small enough that one might run across many larger instances in real world settings. The problem instances given by PATAT had:

- 200 students
- 400 exams
- 10 rooms each with 10 features
- 45 timeslots split evenly between five days

For the PATAT competition, speed was of importance, and contestants were to solve many different instances of this size by running their program on a single processor machine for a limited period of time. The time allowed for each machine was computed by a benchmarking program that took many factors into account, on a Pentium 4 system we were limited to roughly three minutes.

3 Method

We set out to find a strictly combinatorial solution to this problem. We decomposed the problem into several stages, each of which we solve using classic combinatorial optimization methods. Others have previously looked at similar decompositions of the exam scheduling problem, and they have been referred to as cluster methods, and some publications on these methods are [10, 11, 16]. For discussion of cluster methods and other commonly used approaches to solve this problem see [4].

3.1 Find Initial Solution

The goal of our initial phase is to assign events to rooms in timeslots $V_1, V_2, \ldots, V_n$ such that all events are assigned to rooms which can accommodate them, and no student has multiple exams scheduled into the same timeslot. Initially we

Algorithm 1 Timetabling heuristics mainline

```
Given
  number of events $n_e$
  number of features $n_f$
  number of students $n_s$
  number of rooms $n_r$ and size of each $s_i, 0 \leq i < n_r$
  event feature matrix $M^{EF}_{n_e \times n_f}$
    $[M^{EF}]_{ij} = \begin{cases} 1 & \text{event } i \text{ requires feature } j \\ 0 & \text{otherwise} \end{cases}$
  room feature matrix $M^{RF}_{n_r \times n_f}$
    $[M^{RF}]_{ij} = \begin{cases} 1 & \text{room } i \text{ has feature } j \\ 0 & \text{otherwise} \end{cases}$
  student event selection matrix $M^{SE}_{n_s \times n_e}$
    $[M^{SE}]_{ij} = \begin{cases} 1 & \text{student } i \text{ has chosen event } j \\ 0 & \text{otherwise} \end{cases}$
FindInitialSolution()
SatisfyHard()
MinimizeSoft()
```

will call these timeslots *virtual timeslots*, and not restrict their number, or assign them to an actual timeslot with a specific day and time until later. Note that this assignment may or may not represent a feasible schedule. If $n > 45$ the corresponding schedule is not feasible.

Our first step is to construct a student conflict graph G where $V(G) = \{e|\ e \text{ is an exam}\}$, and $E(G) = \{(e_i, e_j)|\ e_i \text{ and } e_j \text{ have at least one common student}\}$. We will also define a weight function $\omega(e) = \delta_G(e)$, where $\delta_G(e)$ is the degree of e in G. The reason this graph is of interest is that since the edges represent common students between two events, any set of events eligible to go in the same timeslot would be a stable set of G. With this in mind, the next step is to greedily choose a maximal stable set S from this graph. We will choose vertices with high weights first and in the case of a tie we will choose randomly from the vertices with highest weights. The vertices we choose will next be matched to assigned to rooms. We choose vertices with high weights first so events that are highly conflicting with other events are given first chance to be assigned rooms.

The next stage of our algorithm is to construct a weighted bipartite event-room graph B where $V(B) = \{S \cup R|\ S \text{ is the stable set chosen from } G \text{ and } R \text{ is the set of rooms available}\}$, and $E(B) = \{(e, r)|\ e$ is an event in S and

r is a room that can accommodate $e\}$. Typically in such a graph $|S|$ will be larger than $|R|$, because $|S|$ was chosen to be maximal in G. This graph is constructed as a tool that will help us match some subset of events from S to rooms in R. We now will let n_r represent the number of rooms, which in our case is 10, and define a waste function for each edge $\alpha(e,r)$ to be the number of features in r *unused* by e. Define the weight of the edges in B as:

$$\omega(e,r) = 2^{n_r+2-\delta_B(e)} + \delta_G(e) - \alpha(e,r)$$

where $\delta_G(e)$ and $\delta_B(e)$ are the degrees of e in G and B. The weights can be thought to represent the priority of matching a particular event to a room. This function was determined through experimentation, but is not random. For example, events with a low degree in B will have a high priority when matching events to rooms because there are few rooms that can accommodate them. We will also give higher priority to events with a high degree in G because such events are incompatible with many other events and we want to get them matched as soon as possible. The waste function α is also included as a way to give priority to event-room combinations with "low waste", or a low number of unused features. Once B is constructed we find a maximum weighted matching. This is a well known problem and in our implementation we used the Hungarian method.

Algorithm 2 Initial solution (FindInitialSolution())

```
Construct event conflict graph $G$
$n_{vt} := 0$
while $|V(G)| \neq 0$ do
    Find a maximal independent set $S \subseteq V(G)$ greedily
    Construct bipartite event room graph $B$
    Find a weighted maximum matching $M$ in $B$
    Assign all events in $M$ to virtual time slot $n_{vt}$ and increment $n_{vt}$
    Delete all events in $M$ from graph $G$
end while
```

A matching in B represents an assignment of events from S to separate rooms. Since S is a stable set in G, this assignment of events to rooms can be assigned to a timeslot. We record this assignment of events to rooms in a virtual timeslot V_i, remove the vertices in G corresponding to events placed in V_i, and repeat this process. We will continue to repeat this process until G is empty. Once this is complete we are left with a valid assignment of all events to rooms within virtual timeslots $V_1, V_2, \ldots, V_n$. If $n \leq 45$ this represents a feasible schedule that satisfies all hard constraints by arbitrarily assigning virtual timeslots to the real timeslots. However, as mentioned earlier, if $n > 45$ we do not yet have a feasible schedule, and the next phase of our algorithm addresses this problem.

3.2 Satisfy Hard Constraints

We enter this phase with a set of virtual timeslots $V_1, V_2, \ldots, V_n$ that contain assignments of events to rooms in a feasible manner. The purpose of this phase is to essentially ”squeeze” the events into fewer timeslots.

Our approach is to find a way to rearrange the events into different rooms and timeslots in order to empty some of the higher indexed timeslots all together. We achieve this by modeling the problem as a flow problem on the following directed graph D. Let $V(D) = \{v| \ v \text{ is a room within any timeslot}\}$ and $E(D) = \{(v_i, v_j)| \ v_i$ contains an event that can be moved into v_j without rendering the resulting schedule infeasible$\}$. To describe more precisely how the edges are chosen, we say that a directed edge goes from v_i to v_j in the case that, if we were to remove an event contained in v_j, an event from v_i could replace it. To see if an event in v_i can be placed in v_j we must check a few things. The room in v_j must accommodate the event in v_i, meaning it must have enough seats and all the required features. We also need to ensure that no student taking the exam in v_i, is also taking another exam in any of the exams scheduled in the same timeslot as v_j. Both of these can easily be checked using information we could record from the previous phase of the problem.

Now consider a directed path in D terminating at a vertex that corresponds to an empty room. If we are to take the event in v_i and move it to v_j for every edge (v_i, v_j) in our path, then this represents a rearrangement of events into different rooms and timeslots. There is one small problem here. If we originally build all the edges of D then taking such a path and rearranging the events accordingly may not always result in a feasible schedule. For example, suppose some event contained in v_i can be moved to v_j, and (v_i, v_j) is in D. Now suppose that suppose some event contained in v_m can be moved to v_n, and (v_m, v_n) is also in D and v_j and v_n are both in timeslot V_1. There is nothing that ensures the events in v_i and v_m have no students in common, so rearranging all the events according to the path may leave the events previously in v_i and v_m in v_j and v_n which are both in V_1 giving us a student conflict, and an infeasible schedule. This problem is easy to avoid, but considering it is important to ensure we do not augment a feasible schedule and convert it into an infeasible one. We will discuss avoiding such infeasible schedules when describing our use of this graph

Recall that our goal of this phase is to rearrange events to require less

Algorithm 3 Hard constraints satisfaction (SatisfyHard())

```
Construct graph $D$
while $n_{vt} > 45$ do
    Choose source $v \in V_i$ where $i > 45$
    run maxflow-mincut algorithm on $D$
    if there is a path $P = v_1, \ldots, v_n$ then
        for each arc $(v_i, v_{i+1}) \in P$ do
            Move event from slot $i$ into slot $i + 1$
        end for
        Recompute number of time slots $n_{vt}$
    end if
end while
```

timeslots. We will accomplish this by doing the following: Construct D as described above, add a new vertex t to D, add directed edges from every vertex corresponding to an empty room in D to t whenever that empty room is in V_i for $1 \leq i \leq 45$, let a vertex containing an event in any timeslot with index greater than 45 be a source, and let t be a sink and apply the max-flow algorithm. As mentioned previously we have to exercise caution in how we create augment along paths in D to avoid rendering the schedule infeasible. In order to avoid such problems we were able to construct the edges in D dynamically, incorporating the extra condition that if we augment on any path leading from the source to any directed edge, it will not produce a conflict. Continue to repeat this process for every event that is not included in the first 45 timeslots, and eventually every event should be squeezed into the first 45 timeslots.

There is no guarantee this will give us a feasible schedule if one exists. However, we may recall that determining if a feasible schedule even exists for a problem instance is NP-complete. However, in practice this worked extremely well for compressing the events into fewer timeslots.

3.3 Minimize Soft Constraints

By now we have virtual timeslots $V_1, V_2, \ldots, V_n$, where $n \leq 45$, and we are left to assign these virtual timeslots to the real timeslots (i.e. Monday 8am). We can also think of this as a reindexing of $V_1, V_2, \ldots, V_n$, in order to minimize the soft constraints.

In the problem description we list three soft constraints: minimization of consecutive exams, minimization of exams written in last timeslot of the day, avoiding students with single exams in a given day. These were given as part of the problem posed in the PATAT competition that we developed this method for. In practice, we saw that the first two of these three were by far the most troublesome, so our method focuses on addressing these and ignores the third soft constraint. We therefore want to order the exams in such a way that attempts to avoid students having more than two consecutive exams and minimizes the number of students taking exams in the last timeslot of the day.

In order to attempt to minimize the number of students with an exam at the end of the day we would essentially extend our previous phase, satisfying the hard constraints. Instead of stopping once we had all exams in 45 timeslots, we would continue the process trying to fit them into fewer and fewer timeslots. In our problem instance, it would be impossible to fit all exams in less than 40 timeslots.

Once we had the exams scheduled into as few timeslots as possible, we sought out to order the timeslots in a manner that would attempt to minimize the number of students with more than two consecutive exams. We formulated this as a pathfinding problem in a hypergraph and solved it with a greedy approach.

Let H be a hypergraph where $V(H) = \{V_i | 1 \leq i \leq 45\}$ and $E(H) = \{(V_i, V_j, V_k) | i, j, k \in V(H)\}$. Notice that the vertex set consists of 45 timeslots, even if all events are scheduled into fewer than 45 timeslots, we will include the empty ones up through 45 in our graph. It is of interest to define a weight on this graph and we will do so as follows, letting $S(V_i)$ represent the set of students attending an event in V_i:

$$\omega(V_i, V_j, V_k) = |\bigcup_{n \in \{i,j,k\}} S(V_n)|$$

For a path of length three or more $(p_0, p_1, \ldots, p_n)$, let the weight of that path be

$$\sum_{i=0}^{n-2} \omega(p_i, p_{i+1}, p_{i+2})$$

Now we can see that finding five paths each of length nine in H with a combined weight of p is the same as finding a five day timeslot arrangement with a penalty from the avoid multiple consecutive exams soft constraint of p.

As mentioned earlier we will only concern ourselves with minimizing this penalty, and the penalty caused by the number of students writing exams in the last timeslot of the day. In order to choose the five paths we do the following: set aside five vertices with low weight to reserve for the last timeslots of they day. Then greedily choose five paths of length five with low weight. Greedily add remaining vertices one by one to any of the paths that will give the least penalty (stopping adding vertices to paths when they reach length eight). Once all paths are of length eight, add the five vertices initially set aside, one to each path, in the way that will incur minimum penalty. All of this is relatively computationally inexpensive, especially because the edge weights can be computed dynamically in an efficient way. For example, once we have a path, in order to compute the cost of adding any of the other loose vertices we can take the intersection of the students in the last two timeslots of the path, and then take the intersection of that with the student set of each of the other timeslots one by one.

Algorithm 4 Soft constraint minimization (MinimizeSoft())

```
Construct the virtual time slot set of students $V_i$
    $V_i = \{j \mid \text{student } j \text{ is busy during time } i\}$
Construct $\mathcal{V} = \{V_1, V_2, \ldots, V_{n_{vt}}\}$
Set aside five $V_i$ of minimal student count
Construct initial ordered list $D_k = (V_{i_k}, V_{j_k}), 1 \leq k \leq 5$, one per day
    where the pairs $(V_{i_k}, V_{j_k})$ have minimal intersection among all pairs
$\mathcal{V} = \mathcal{V} \setminus \{V_{i_1}, V_{j_1}, \ldots, V_{i_5}, V_{j_5}\}$
while $|\mathcal{V}| \neq 0$ do
  Choose $V_i$ of minimal intersection with head or tail of each a list, say
  $D_k$.
  Add $V_i$ to $D_k$ and delete from $\mathcal{V}$
end while
Add the originally saved five $V_i$, one per list $D_k$, minimizing penalty asso-
ciated with being last time slot of day
```

Once we have found all five of these paths of length nine, we arrange them into each of the five days, putting whichever of the head or tail has less students in it into the last timeslot of the day. This will be our final timeslot.

[QUESTION?: DID WE IMPLEMENT 2, 3-OPS HERE TO FURTHER IMPROVE THE SCHEDULE?]

4 Implementation and Results

As previously mentioned, the motivation for developing this method was in part to compete in an international timetabling competition organized by PATAT. For more details about the competition we refer the reader to the competition website [http://www.idsia.ch/Files/ttcomp2002/]. We implemented our algorithm in C. Our implementation was able to solve all instances within approximately $1/50^{th}$ of the time issued. Our implementation also was made up of different smaller programs for each phase, and required file read and write time between each phase, so a fully integrated version could be potentially much faster. For the competition, we would use our remaining time to rerun the program multiple times. Although solutions usually had similar penalties, the randomness in choosing stable sets in the initial phase made additional runs produced different solutions.

The format of the competition was that we were issued ten instances upon entry to test, and then within the last week of the deadline we were issued ten fresh instances. We were required to reproduce results if needed. Of all the teams that entered, we were one of only 22 to successfully complete all the problem instances. We will give our results for the 20 instances.

First we give some data about the instances:

Instance	Students	Rooms	Room Equipment	Ave Exam/Student

The following table gives the runtime for each instance and the number of violations for each soft constraint.

Instance	Runtime	Last Timeslot	3 in-a-row	One per day

The problem instances and results of selected other participants are available for download at the competition webpage [http://www.idsia.ch/Files/ttcomp2002/].

5 Conclusions

In this paper we gave a specific student exam scheduling problem, and a strictly combinatorial approach to solving it. Although we did not win the competition in which this problem was posed, the leading teams did not use a strictly combinatorial approach. The speed of our method is a strong point. Unlike the hill-climbing, local search and genetic style algorithms that would need all the available time to reach a good solution, we were able to reach our solution in a matter of a seconds as opposed to minutes. Looking back at the combinatorial approach of each stage of our algorithm, it can also be seen that increasing parameters such as the number of students or exams, would not greatly increase the runtime. Therefore our method would be promising for larger scale problems in a real world setting.

References

[1] Victor A. Bardadym. Computer-aided school and university timetabling: The new wave. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling (PATAT, Edinburgh, U.K., Aug./Sept. 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*, pages 24–45. Springer-Verlag, Berlin Heidelberg New York, 1996.

[2] E.K. Burke, Y. Bykov, and S. Petrovic. A multicriteria approach to examination timetabling. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III (PATAT, Konstanz, Germany, Aug. 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 118–131. Springer-Verlag, Berlin Heidelberg New York, 2001.

[3] E.K. Burke, J.P. Newall, and R.F. Weare. A memetic algorithm for university exam timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling (PATAT, Edinburgh, U.K., Aug./Sept. 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer-Verlag, Berlin Heidelberg New York, 1996.

[4] Michael W. Carter and Gilbert Laporte. Recent developments in practical examination timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling (PATAT, Edinburgh, U.K., Aug./Sept. 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*, pages 3–21. Springer-Verlag, Berlin Heidelberg New York, 1996.

[5] M.W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34:193–202, 1986.

[6] Philippe David. A constraint-based approach for examination timetabling using local repair techniques. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II (PATAT, Toronto, Canada, Aug. 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*, pages 169–186. Springer-Verlag, Berlin Heidelberg New York, 1998.

[7] W. Erben. A grouping genetic algorithm for graph colouring and exam timetabling. In Edmund Burke and Wilhelm Erben, editors, *Practice*

and Theory of Automated Timetabling III (PATAT, Konstanz, Germany, Aug. 2000, Selected Papers, volume 2079 of *Lecture Notes in Computer Science*, pages 132–156. Springer-Verlag, Berlin Heidelberg New York, 2001.

[8] L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III (PATAT, Konstanz, Germany, Aug. 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer-Verlag, Berlin Heidelberg New York, 2001.

[9] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Edmund Burke and Patrick DeCausmaecker, editors, *Practice and Theory of Automated Timetabling IV (PATAT, Gent, Belgium, Aug. 2002, Selected Papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 262–275. Springer-Verlag, Berlin Heidelberg New York, 2003.

[10] D. Johnson. Timetabling university examinations. *Journal of the Operational Research Society*, 1990.

[11] V. Lofti and R. Cerveny. A final-exam-scheduling package. *Journal of the Operational Research Society*, 1991.

[12] T. Nepal, S.W. Melville, and M.I. Ally. A brute force and heuristics approach to tertiary timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II (PATAT, Toronto, Canada, Aug. 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*, pages 254–265. Springer-Verlag, Berlin Heidelberg New York, 1998.

[13] Ben Paechter, R.C. Rankin, and Andrew Cumming. Improving a lecture timetabling system for university-wide use. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II (PATAT, Toronto, Canada, Aug. 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*, pages 156–165. Springer-Verlag, Berlin Heidelberg New York, 1998.

[14] David C. Rich. A smart genetic algorithm for university timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling (PATAT, Edinburgh, U.K., Aug./Sept. 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*, pages 182–197. Springer-Verlag, Berlin Heidelberg New York, 1996.

[15] Peter Ross, Emma Hart, and Dave Corne. Some observations about ga-based exam timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II (PATAT, Toronto, Canada, Aug. 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*, pages 115–129. Springer-Verlag, Berlin Heidelberg New York, 1998.

[16] G.M. White and P.W Chan. Towards the construction of optimal examination timetables. *INFOR*, 17:219–229, 1979.

[17] G.M. White and B.S. Xie. Examination timetables and tabu search with longer-term memory. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III (PATAT, Konstanz, Germany, Aug. 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 85–103. Springer-Verlag, Berlin Heidelberg New York, 2001.