Computational Optimization and Applications, 16, 279–294, 2000
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

Scheduling Classes on a College Campus

PERRY FIZZANO* pfizzano@ups.edu
STEVEN SWANSON sswanson@ups.edu
Department of Mathematics and Computer Science University of Puget Sound, Tacoma, WA 98416, USA

Received September 11, 1997; Accepted October 8, 1998

Abstract. We consider the problem of scheduling a set of classes to classrooms with the objective of minimizing the number of classrooms used. The major constraint that we must obey is that no two classes can be assigned to the same classroom at the same time on the same day of the week. We present an algorithm that produces a nearly optimal schedule for an arbitrary set of classes. The algorithm's first stage produces a packing of classes using a combination of a greedy algorithm and a non-bipartite matching and the second stage consists of a bipartite matching.

First we show that for one variant of the problem our algorithm produces schedules that require a number of classrooms that is always within a small additive constant of optimal. Then we show that for an interesting variant of the problem the same algorithm produces schedules that require a small constant factor more classrooms than optimal. Finally, we report on experimental results of our algorithm using actual data and also show how to create schedules with other desirable characteristics.

Keywords: timetabling, bipartite matching, assignment, approximation algorithm

1. Preliminaries

Scheduling classes at a University can be a difficult problem depending on the type of constraints that the schedule must obey. We consider the problem of assigning an arbitrary set of classes to a collection of classrooms with the objective of minimizing the number of classrooms used. This problem was motivated by our University's registrar. The registrar's office wanted to determine how efficiently the University was using its classroom space. The input to the problem is a set of classes such that each class meets on a predefined subset of the five days of the work week for one or two hours. Furthermore, each class must start at the top of an hour. The major constraint that we must obey is that no two classes can be assigned to the same classroom at the same time on the same day of the week. We present an algorithm that assigns the classes to classrooms in a nearly optimal fashion.

A classic problem in the literature closely related to our problem is known as the timetabling problem. There have been many variants of the timetabling problem (many specifically dealing with a form of classroom scheduling) explored by researchers. Some papers have shown how to formulate the problem as a mathematical programming problem [4] while others have proposed heuristic approaches to find suitable solutions [5]. An article by Carter and Tovey [2] settled some of the prior claims in the literature relating to

*Author to whom correspondence should be addressed.

the hardness of classroom assignment. They resolved the complexity of many variants of the problem by giving hardness proofs or polynomial time algorithms. However, none of the variants they consider directly corresponds to our problem. One difference between our work and previous work is that we are giving full flexibility to the scheduler to minimize the number of classrooms. Typical examples of the classroom scheduling problem do not give such flexibility; teachers and departments have preferences or requirements for certain rooms or times that are considered when making the schedule. We discuss at the end of the paper how to modify our algorithm to take into account various other constraints that a university might like to include in the classroom assignment problem. Another difference between our work and other timetabling related research is that we employ the techniques of approximation algorithms [10] to give worst case bounds on the performance of our algorithm.

Our problem is also related to problems in the field of parallel machine scheduling, specifically problems in that area that consider an underlying communication network. The problem we consider differs from some previously considered scheduling problems in that there are three dimensions to our problem that must be considered: classroom, time, and day of the week. It is the presence of this third parameter that differentiates our problem from problems such as the stable marriage problem [8, 9] or scheduling unit-sized jobs on a network of parallel machines with communication delays [3, 6, 11].

We proceed to explain the connection between our problem and a variant of parallel machine scheduling as the methods of solution are related. In parallel machine scheduling with communication delays only one unit-sized job can be assigned to each machine at any given time and each job takes one time unit. Jobs may be moved from one machine to another machine but moving requires time proportional to the distance between the two machines. One proposed solution [6] entails setting up the scheduling problem as an assignment or a bipartite matching problem (see chapter 12 of [1] for more about bipartite matchings). A bipartite graph $G = (V, E)$ is a graph in which the vertex set $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$ and $(i, j) \in E$ implies that either $i \in V_1$ and $j \in V_2$ or $i \in V_2$ and $j \in V_1$. To phrase the parallel machine scheduling problem with communication delays as a bipartite matching problem let each vertex in V_1 represent one of the unit-sized jobs, and let each vertex in V_2 represent a machine/time pair. An edge is placed between vertex $i \in V_1$ and vertex $j \in V_2$ if the job represented by i could reach the machine represented by vertex j by the time represented by vertex j. A matching in which each job is matched to some machine at a particular time represents a valid schedule since each machine only processes one job at a time and each job is processed on exactly one machine.

Now consider the problem in setting up the class scheduling problem as a bipartite matching problem where V_1 represents the set of classes to be scheduled and each vertex in V_2 represents a classroom/time pair for the entire week. If edges are simply placed between classes and classrooms that are compatible (i.e. the maximum enrollment of the class would fit into the classroom) a maximum bipartite matching will not necessarily result in a good schedule because multiple classes can use a classroom at the same time as long as they use it on different days of the week. That is, each classroom/time pair could be assigned more than one class as long as the set of days each of those classes meet are disjoint. It is this third dimension to our problem that we must deal with in order to get good schedules.

As a result, we broke our algorithm for the class scheduling problem into two major parts. The first part of our algorithm creates a set of *blocks* of classes. Each block represents two hours of the day for five days of the week. Each block is filled with a set of classes that can coexist in some two hour slot for the week. We could, for example, place two classes that meet for one hour on Monday, Wednesday and Friday in the same block as a single two-hour class that meets on Tuesday and Thursday. By creating this set of blocks we have essentially reduced the problem to a two-dimensional problem.

The second part of the algorithm assigns the blocks created in part one to classroom/time pairs. We perform this assignment by performing a maximum-cardinality bipartite matching similar to what we described above. In the construction we use, each vertex in V_1 represents a block while V_2 represents the set of classroom/time pairs. An edge is present between a block and a classroom/time pair if all the classes in the block would fit in the room. In the experimental section of the paper we place weights on the edges to represent the degree of compatibility between a block and a classroom/time pair.

1.1. Definitions and overview

A *two-hour block* is a 2×5 array of cells representing two hours on each of the five days of the work week, and a *one-hour block* is a 1×5 array of cells representing one hour on each of the five days of the work week. Note that we do not say which hour or which room each block represents beforehand, that will be decided in the second part of the algorithm. An *n-class* is a class that meets n days out of the week. We assume that a class may meet on any predetermined subset of the five days of the week for either one or two hours per meeting. We require that each class meeting start at the same time throughout the week.

A *k-block* is a block that has at least one cell in k of its columns filled and we say that the *fullness* of a k-block is k. We graphically represent a two-hour block as shown in figure 1. This block contains two classes, A and B. A is a 3-class and meets for two hours on Monday, Wednesday and Friday, while B is a 2-class and meets for one hour on Tuesday and Thursday. The cells containing a dash indicate that no class is contained in that part of the block.

We say a *conflict* occurs between two classes, A and B, if they could not coexist in a single block. If two classes do not conflict they are *compatible*.

It turns out that our registrar is only interested in schedules in which all two-hour classes start at even hours (8:00, 10:00, etc.). We call this problem the *boundary-respecting* problem. However, it is of mathematical interest to allow two-hour classes to start on odd hours as well. We will refer to this version of the problem as the *non-boundary-respecting* problem and discuss that variant in Section 3.1.

A	B	A	B	A
A	-	A	-	A

Figure 1. A 5-block with two classes in it.

Notice, that in the boundary-respecting problem, schedules can be broken up into two-hour blocks like those described above since no single class meeting (whether it be a one-hour class or a two-hour class) will span the start of an even hour. This observation, combined with the constraint that all classes are either one or two hours long, reduces our problem to building the smallest number of two-hour blocks possible.

Thus, our objective is to take a set of classes and pack them into as few two-hour blocks as possible. We will say that twice the number of blocks is the *length* of the schedule (e.g. we say that a set of classes that requires 5 two-hour blocks is of length 10 because it spans 10 hours). We define $\text{Opt}(S)$ as the minimum possible boundary-respecting schedule length for a set, S, of classes.

The layout of the paper is as follows. In the next section we prove two simple lower bounds on the length of the optimal schedule for a set of classes. In Section 3 we present the algorithm and provide an analysis for the boundary respecting problem that shows the algorithm builds schedules that use nearly the minimum number of classrooms. In that same section we also provide an analysis of the same algorithm for the non-boundary-respecting problem. Finally, in Section 4 we report on the performance of our algorithm on actual data from the University of Puget Sound.

2. Lower bounds

In this section we make some simple observations about how a set of classes imposes lower bounds on the length of the optimal schedule. We assume that all classes in the set, S, meet for h hours per day. Also, let C_i be the number of classes that meet on day i for $i \in \{1..5\}$ (i.e. Monday is 1, Tuesday is 2, etc.).

Lemma 1. $\text{Opt}(S) \geq h \max_{i \in 1..5} C_i$.

Proof: Consider a day i for which $C_i = j$. The classes that occupy day i need at least hj hours of classroom time on that day, and therefore the schedule must be at least hj hours long. This is true for all i and particularly the i for which C_i is largest. □

Let the number of k-classes in S be denoted by D_k.

Lemma 2. $\text{Opt}(S) \geq h(D_3 + D_4 + D_5)$.

Proof: Assume that some pair of 3, 4, or 5-classes did not conflict and thus could coexist on the same line of a block. For this to happen there would have to be more than 5 columns in that block. This is a contradiction, so each 3,4, and 5-class must occupy h lines itself. □

3. Algorithm and analysis

Our algorithm consists of two major functions. The first, GROUP_ALL_CLASSES, calls the second function, GROUP_CLASSES. GROUP_CLASSES combines a set of classes into the minimum number of two-hour blocks possible. It works correctly when the classes are of the same length (either one hour or two hours) and are sorted by non-increasing fullness with ties broken arbitrarily. However, we only use GROUP_CLASSES on two-hour long classes. To ease exposition we say that each class is initially placed in a block by itself.

GROUP_CLASSES takes a block, c, from the top of the list and starts comparing it with the other blocks in the list in order. If it finds a block, g, that is compatible with c it merges the block g into the block c and removes g from the list. It continues down the list looking for a block that is compatible with the the new c. When it reaches the end of the list, the algorithm places c in a list of processed blocks and repeats the procedure. The algorithm proceeds like this until it has processed all the $5, 4$, and 3-blocks in the list.

Then the algorithm considers the remaining blocks and builds a graph, $G = (V, E)$, where

$$V = \{b \mid \text{fullness}(b) = 2\}$$

and

$$E = \{(b, c) \mid b \text{ is compatible with } c\}.$$

The algorithm performs a maximum cardinality matching on G. It combines each matched pair of 2-blocks into a single block and inserts them into the sorted list of blocks, preserving the ordering of fullness. The algorithm then proceeds as before; it tries to find blocks compatible with the 4-blocks produced during the matching and it proceeds until all the blocks have been processed.

GROUP_ALL_CLASSES, creates a schedule from a set, S, of blocks. First, it examines the one-hour long blocks. If two one-hour long blocks meet on exactly the same days of the week, the algorithm places them in one two-hour block (figure 2). Then it runs GROUP_CLASSES on these two-hour blocks along with the original two-hour blocks. Next, it combines the remaining one-hour 2-blocks with the one-hour 3-blocks and the one-hour

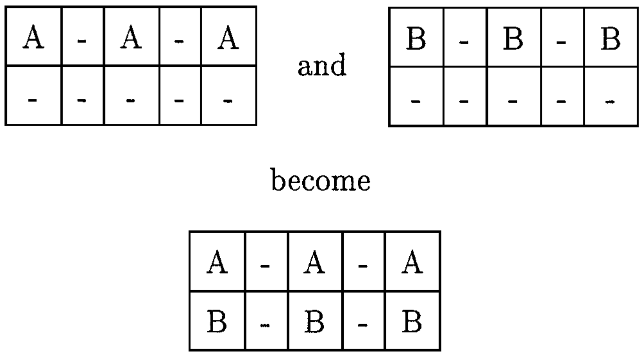

Figure 2. Two similar one-hour blocks combined into a single two-hour block.

1-blocks with the one-hour 4-blocks. Finally, it adds the left over one-hour 5-block if it exists.

GROUP_ALL_CLASSES produces schedules that have a length less than Opt(S) + 15 hours and it runs in $O(n^{2.5})$ where n is the number of classes to be scheduled. The matching used [13] in GROUP_CLASSES is the bottleneck in terms of the running time.

Our algorithm produces boundary-respecting schedules that are no more than 15 hours longer than optimal. Recall, boundary-respecting schedules are schedules that can be broken into two-hour blocks without splitting any two-hour classes.

In order to demonstrate the correctness of our algorithm we first show that GROUP_CLASSES schedules a set of classes of one length (one or two hours) optimally (Lemma 3). Then we will show that GROUP_ALL_CLASSES produces schedules that require fewer than Opt(S) + 15 hours (Theorem 1) for an arbitrary set of classes, S.

Lemma 3. *For a set*, S, *of classes of one length*, h, GROUP_CLASSES *produces a schedule of length* Opt(S).

Proof: The proof is by induction on the number of blocks to be scheduled. Recall that each class is initially placed in its own block.

Basis: We can schedule one block optimally.

Inductive Hypothesis: Assume that the algorithm, A, creates an optimal schedule, $A(S)$, for a set, S, of $k \geq 1$ blocks sorted in order of non-increasing fullness. Then it will also create an optimal schedule, $A(S')$, for a set, S', of $k + 1$ blocks sorted in the same order.

We can assume without loss of generality that the $k + 1$st block, a, is no more full than any of the other blocks in S. This means that a is the last class in the sorted list. There are a number of cases to consider.

1. a is a 3, 4, or 5-block. This implies that there are $(k + 1)$ 3, 4, and 5-blocks in the list. These blocks constitute a schedule that is $h(k + 1)$ hours long. By Lemma 2 this schedule is optimal.
2. a is a 2-block
   (a) a is combined with a 3-block, $b \in S$, that was not combined with any block in schedule $A(S)$. Since the length of $A(S')$ will be the same as the length of $A(S)$, $A(S')$ must be optimal since our lower bounds only increase when we increase the number of classes.
   (b) a is not combined with a 3-block. Thus, a is one of the 2-blocks involved in the maximum cardinality matching.
      i. If the cardinality of this matching increases by one, then $A(S')$ will the same length as $A(S)$. Therefore, $A(S')$ will be optimal.
      ii. Otherwise, the matching will not combine a with another block. Since the matching is optimal, there is no way to combine a with another block and produce a shorter schedule. In this case $A(S')$ will also be optimal.

3. a is a 1-block that meets on day i. If there is a space in $A(S)$ on day i, a will be placed there and the schedule length will not increase thus the schedule length is still optimal. Otherwise, a will require its own block, and $A(S')$ will be h hours longer than $A(S)$. By Lemma 1, $A(S')$ will be optimal. □

We say that two blocks are of the same *type*, if they are the same length and meet on exactly the same days of the week. Note that there are $2^5 = 32$ different subsets of days of the week. Since, we are not concerned with classes that meet on zero days during the week, there are 31 types of classes and thus 31 types of blocks at the start of the algorithm.

Let $M(S)$ represent the maximum cardinality matching of a set, S, of 2-blocks. Each pair of vertices that are matched will contribute one to the total number of blocks used for the classes in S. Each vertex that is unmatched will also contribute one to the number of blocks used. Let $B(S)$ denote the number of blocks that will be used for the set, S, of classes. We use $S + S$ to denote the set of blocks produced by duplicating each block in S.

Lemma 4. *If $B(S) = k$ then $B(S + S) = 2k$.*

Proof: For two blocks, x and $y \in S$, there will be two additional blocks, x^* and y^* identical to x and y respectively, in $(S + S)$.

If x and y are matched in $M(S)$ then in $M(S + S)$ x could be matched with y, and x^* could be matched with y^*. Each such pair of blocks will result in two matches in $M(S + S)$ while requiring only one in $M(S)$ thus $B(S + S) = 2B(S)$.

If x and y are incompatible blocks (thus they are not matched in $M(S)$) then x will be incompatible with x^* because they meet on the same days of the week, and x will also be incompatible with y^* because y^* is identical to y. By the same argument, y will be incompatible with y^* and x^*. These classes will require four blocks in $M(S + S)$ while requiring two in $M(S)$. Again, this implies that $B(S + S) = 2B(S)$. □

We will now show that for a restricted set of instances our algorithm produces an optimal schedule. This restricted set of instances requires that the set of classes contain an even number of each type of one-hour class and any number of two-hour classes. In the following lemma we use I to denote an instance of such a set of classes.

Lemma 5. GROUP_ALL_CLASSES *produces an optimal schedule for a restricted set, I, of classes.*

Proof: Our restricted set of classes, I, contains an even number of each type of one-hour class. For each of these pairs of one hour classes place one in a set A and the other in a set B. The set, I, also contains two-hour classes. Split each of these two-hour classes into two one-hour classes. Place one half in A and the other half in B. We denote the individual members of set A by a_i and similarly we denote the members of set B by b_i. Note that A is identical to B and moreover, a_i is identical to b_i for all i.

Running GROUP_CLASSES on A and B will produce two identical schedules, S_A and S_B. By Lemma 3 both of these schedules will be optimal. Now, we interleave the blocks from S_A and S_B to create two-blocks so that the two pieces of the two-hour classes that we split will be reunited. The resulting schedule is optimal because both of our lower bounds (given by Lemma 1 and Lemma 2) have doubled. Also note that by Lemma 4, the number of blocks required by the matching will also double. This means the schedule created merging these two schedules will also be optimal. □

Let S be an arbitrary set of classes, and let $L(S)$ be the length of the schedule that GROUP_ALL_CLASSES produces for S.

Theorem 1. $L(S) \leq \text{Opt}(S) + 15$.

Proof: GROUP_ALL_CLASSES first builds as many two-hour blocks as possible by combining one-hour blocks of the same type. Then it uses GROUP_CLASSES to schedule the two-hour blocks. By Lemma 5 this schedule will be optimal for this set of classes.

There will be at most one left over one-hour 5-block. Adding this to the end of the schedule will maintain the schedule's optimality (Lemma 1).

There will be at most 10 left over one-hour 3-blocks, 10 left over one-hour 2-blocks, 5 left over one-hour 4-blocks, and 5 left over one-hour 1-blocks. These can be scheduled in 15 hours by combining 2-blocks with 3-blocks, and the 4-blocks with the 1-blocks.

It is possible that we could have arranged the schedule so that the left over classes would fit into it. Therefore, the schedule we produce might be as much as 15 hours longer than an optimal schedule. □

Now that we have built nearly the minimum number of blocks we can just assign each block to some room for two hours for the whole week. We assign each block to a room/time pair (where each time is actually a two hour time span) by using a bipartite matching where each vertex in V_1 represents one block and each vertex in V_2 represents a classroom for two hours for the whole week. In order to use the fewest rooms possible the bipartite graph used in this matching should be complete (each block should have an edge to each room/time pair). Such bipartite matchings are trivial to perform. However, in practice, this part of the algorithm will be done using a less than complete bipartite graph to account for the fact that some classes can only meet in certain classrooms because of class size, audio-visual requirements, etc. We will discuss the practical application of this algorithm in Section 4.

Finally, notice how the last theorem in conjunction with the bipartite matching just described relates to our objective of minimizing the number of classrooms used. If the schedule is within 15 *hours* of optimal then that implies that the schedule is within 2 *classrooms* of optimal since one classroom can hold eight hours of class if the schedules run from 8 am to 4 pm.

3.1. An approximation bound for non-boundary-respecting schedules

Recall, that the *non-boundary-respecting* scheduling problem allows two hour classes to start at any hour, not just even hours as we have assumed so far. Thus, non-boundary-respecting schedules allow for two new ways for a group of classes to fit together. The first is an *overlap*. An overlap occurs when a two-hour event starts at time $2i + 1$ for any integer i. We will say that this overlap occurs at hour $2i + 1$. The second is an *interlock*. An interlock occurs when there exists an hour in which one two-hour event is beginning and another two-hour event is ending. Notice that if an interlock exists, then an overlap exists as well. If there were no overlaps (and therefore no interlocks) then we would have a boundary-respecting schedule. Thus, the presence of an overlap is necessary and sufficient to make a schedule non-boundary-respecting.

Another important structure in non-boundary-respecting schedules is the *chain*. A chain, C, is a set of consecutive hours, $C[0]$, $C[1]$, $C[2], \ldots, C[n-1]$, such that hours, $C[1]$, $C[2], \ldots, C[n-2]$ contains an interlock. We say the length of C is n.

We now use $\mathrm{Opt}_{nbr}(S)$ to denote the optimal length schedule for the *non-boundary-respecting* problem and $\mathrm{Opt}_{br}(S)$ for optimal length schedule for the boundary-respecting problem. We still use $L(S)$ to denote the length of the schedule produced by GROUP_ALL_CLASSES.

Theorem 2. $L(S) \leq \frac{3}{2}\mathrm{Opt}_{nbr}(S) + 15$.

Proof: We will show that a non-boundary-respecting schedule, A, of a set of events, S, can be transformed into a boundary-respecting schedule by removing all the chains and overlaps in A. We also demonstrate that the length of the transformed schedule is no more than $3/2$ the length of A.

We will remove each chain one by one from the top of the schedule to the bottom. To remove a chain, C, (figure 3(a)) we perform the following transformation repeatedly. We move all the classes in the hours that the chain occupies to the top of the schedule (so the chain starts on hour 0, an even hour). Then we move the two-hour classes that start at $C[1]$ so they start at $C[0]$. However, this move might cause a conflict with one-hour events that start at $C[0]$. If so, then we remove those one-hour events and put them at the end of the schedule in their own block (figure 3(b)). This transformation removes the interlock at $C[1]$ and leaves a valid schedule. Notice that the chain that remains is two hours shorter. We repeat the transformation with the interlocks at $C[2i + 1]$ for all integers i such that $0 \leq i \leq \lfloor (n-1)/2 \rfloor$ where n is the length of the chain (figure 3(c)).

Now that we have removed overlaps that are part of interlocks, we will remove overlaps that are not part of interlocks. We remove overlaps one by one from the beginning of the schedule to the end. Consider the first overlap and say it occurs at hour k (recall, this means the two-hour event occupies hour k and hour $k + 1$). We move all the classes in hour k and hour $k + 1$ to the beginning of the schedule (figure 4). This move reschedules every event in hours 0 to k two hours later. Since there were no overlaps prior to hour k before the move there will be none prior to hour $k + 2$ after the move. Thus, the number of overlaps decreases every time we make such a move. Also notice that removing an overlap does not increase the length of the schedule, all it does is renumber the hours that events are scheduled.

If the optimal non-boundary-respecting schedule for a set, S, of classes is $\mathrm{Opt}_{nbr}(S)$ hours long it could contain a chain of length $\mathrm{Opt}_{nbr}(S)$. The chain would require $\lfloor (\mathrm{Opt}_{nbr}(S) - 1)/2 \rfloor$ transformations to turn it into a boundary-respecting schedule. Each application of the transformation shortens the length of the chain by two and increases the length of the schedule by at most one. Each transformation that removes an overlap that is not part of an

a. Before the transformation

E	B	E	B	E
A	B	A	B	F
A	C	A	G	C
H	C	D	D	C
-	-	D	D	-

b. After the transformation to remove the interlock at hour 1

A	B	A	B	-
A	B	A	B	F
-	C	-	G	C
-	C	D	D	C
H	-	D	D	-
E	-	E	-	E

c. After the transformation to remove the interlock at hour 3

A	B	A	B	-
A	B	A	B	F
-	C	D	D	C
-	C	D	D	C
H	-	-	-	-
E	-	E	-	E
-	-	-	G	-

Figure 3. Removing a chain from *A*.

Before the transformation:

A	B	A	B	B
C	F	C	F	E
C	D	C	D	D

After the transformation:

C	F	C	F	E
C	D	C	D	D
A	B	A	B	B

Figure 4. Removing an overlap at the boundary between hours 1 and 2.

interlock does not increase the length of the schedule. Therefore, the maximum numbers of hours that could be added to the schedule by transforming it from a non-boundary-respecting schedule to a boundary-respecting schedule is $\lfloor(\text{Opt}_{nbr}(S) - 1)/2\rfloor$. Thus, there exists a boundary-respecting schedule that is $\text{Opt}_{nbr}(S) + \lfloor(\text{Opt}_{nbr}(S) - 1)/2\rfloor \le \frac{3}{2}\text{Opt}_{nbr}(S)$ hours long.

Since GROUP_ALL_CLASSES produces schedules that have a length that is no more than $\text{Opt}_{br}(S) + 15$ and there exists a boundary-respecting schedule that is $\frac{3}{2}\text{Opt}_{nbr}(S)$ hours long we conclude that GROUP_ALL_CLASSES produces schedules that have a length no more than $\frac{3}{2}\text{Opt}_{nbr}(S) + 15$ □

4. Experimental results

The motivation for this project was to give the University of Puget Sound registrar an idea of how efficiently the university utilizes classroom space. The registrar wanted answers to two types of questions: First, if classes meet only during certain hours of the day (e.g. from 8 am to 4 pm), how many classrooms does the university need to hold the classes it currently offers? Second, under the same time constraints what percentage of the classes could be scheduled between 10 am and 2 pm with the current set of classrooms on campus? The registrar was interested in schedules for two different time periods: 8 am to 4 pm and 8 am to 6 pm.

4.1. Building actual schedules

Building real world schedules is a two step process. First, we build the two-hour blocks. Second, we assign each two-hour block to a classroom/time pair.

In order to ensure that classes are placed in rooms large enough to hold them, we need to be careful about how we build the two-hour blocks. Specifically, since all the classes in a block are scheduled in a classroom that can hold the largest class, the classes in a single block should have similar maximum enrollments. Otherwise, we might use the large classrooms inefficiently because they would be occupied by small classes. To guard against this we divide the classes into groups based on their maximum allowed enrollment. Then we build two-hour blocks out of the classes in each group.

Building two-hour blocks with GROUP_ALL_CLASSES is impractical because the $O(n^{2.5})$ algorithm [13] for maximum cardinality non-bipartite matching is extremely complicated.

We developed a simpler algorithm called QUICK_GROUP that is less complicated and performs nearly optimally in practice. It starts by sorting the classes by length (two-hour classes come before one-hour classes) and break ties based on fullness. Then it uses the same greedy strategy as GROUP_CLASSES. The primary differences between GROUP_ALL_CLASSES and QUICK_GROUP are the absence of the non-bipartite matching and the fact that we do not initially combine identical one-hour classes into two hour-blocks. QUICK_GROUP is simple and very fast in practice. It runs in $O(n^2)$ (where n is the number of classes), and we can implement it in about 100 lines of C code.

The number of blocks QUICK_GROUP produces can, for some sets of classes, be at least 3/2 times optimal. In particular, for the blocks in figure 5 QUICK_GROUP will produce three blocks when only two are needed. This example scales if we replace each block with two copies of itself. Fortunately, QUICK_GROUP performed almost optimally for our data sets (see figure 6).

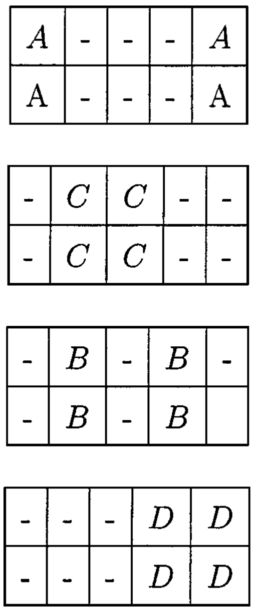

Figure 5. A poor set of classes to run QUICK_GROUP.

Data Set	Lower Bound	QUICK_GROUP before dividing by size	QUICK_GROUP after dividing by size
fall 95	158	158	168
spring 96	167	167	169
fall 96	156	156	158
spring 97	129	129	138

Figure 6. Number of blocks needed for each semester.

After we divide the classes by size and use QUICK_GROUP to build two-hour blocks, we use a maximum cardinality bipartite matching to build the final schedule. To do this we first build a bipartite graph. The first set of vertices represents the two-hour blocks. The second set represents the set of all classroom/time pairs. We place an edge between a block of classes and a classroom/time pair if all the classes in the block are small enough to fit in the classroom. We add weights to the edges to manipulate the distribution of classes throughout the day. To ensure that as many classes as possible will meet during a certain period, we simply increase the weights of the edges involving classroom/time pairs during that period. In particular, we increased the weight on edges incident with nodes representing classroom/time pairs between 10 am and 2 pm.

The most efficient algorithm for weighted maximum cardinality bipartite matching runs in $O(nm)$ for n vertices and m edges [7]. We used the LEDA library's implementation of this algorithm [12].

4.2. Evaluating the schedules we produced

We used our algorithm to build schedules for four semesters of classes at the University of Puget Sound. The class lists we used did not include classes for departments that controlled their own room assignments, classes with specialized needs (e.g. physical education), or classes that, for some other reason, the registrar does not schedule directly. Similarly, the classrooms we used for scheduling were only those that the registrar controls. Since the classes that the registrar schedules are almost always scheduled in the rooms the registrar controls, our task is similar to the registrar's.

We used Lemmas 1 and 2 to calculate lower bounds on the number of blocks needed for each set of data. Then we used QUICK_GROUP to build blocks out of the classes before and after dividing them by size (figure 6). Note that QUICK_GROUP performed nearly optimally even on the divided data sets.

Dividing the classes by size did not significantly increase the number of rooms used on our data set. In the worst case we might need an extra room for every extra two-hour block. In the best cases, we use about two more two-hour blocks (and possibly rooms) than

Data Set	# of classrooms (8:00 to 6:00)	# of classrooms (8:00 to 4:00)
fall 95	38	46
spring 96	38	43
fall 96	37	41
spring 97	29	36

Figure 7. Classrooms needed for each set of data.

necessary. For the Fall 95 data, however, splitting up the classes added 10 two-hour blocks to the schedule. Since each room can only hold five two-hour blocks if classes meet from 8 am to 6 pm and four if they meet from 8 am to 4 pm, our Fall 95 schedule would require at least two or three extra rooms respectively.

To answer the registrar's first question: "How may classrooms does the University need to hold the classes it currently offers?", we scheduled the sets of classes in fewer and fewer rooms until they no longer fit. The results are shown in figure 7. The University currently uses 51 rooms to schedule the classes in our data sets (and they schedule most classes between 8 am and 4 pm although there are exceptions). Our schedules required between 29 and 38 classrooms for the 8 am to 6 pm schedules and between 36 and 46 rooms for the 8 am to 4 pm schedules.

We can answer the second question: "How many classes can be scheduled between 10 am and 2 pm without increasing the current number of classrooms?", by analyzing the schedules we produced using all 51 classrooms. Figures 8 and 9 show the number of classrooms in

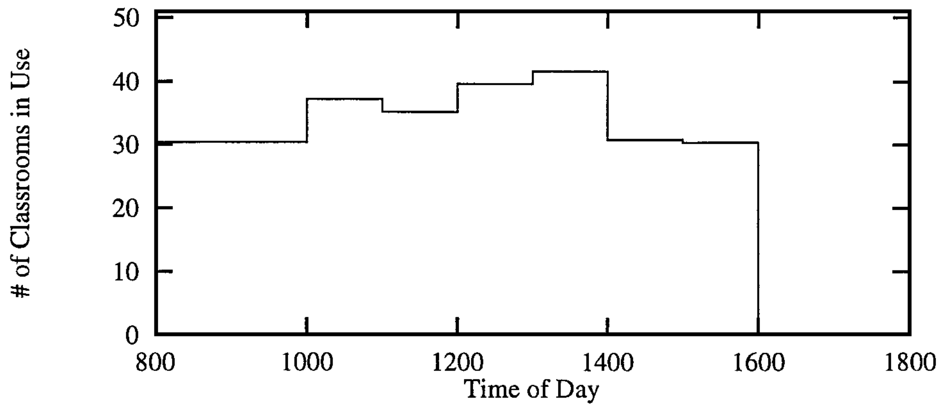

Figure 8. Class distribution Fall '95, 51 classrooms, 8 am to 4 pm.

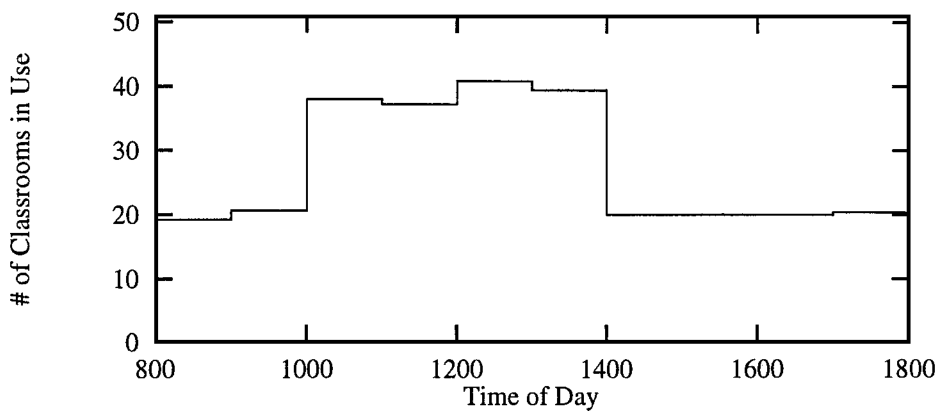

Figure 9. Class distribution Fall '95, 51 classrooms, 8 am to 6 pm.

use throughout the day for the Fall '95 classes scheduled between 8 am and 4 pm and 8 am and 6 pm, respectively, using all 51 classrooms. In both cases we were able to schedule the majority of the classes between 10 am and 2 pm.

The figures show that we never use all of the classrooms. However, the weights in the bipartite matching guarantee that no more classes can be scheduled between 10 am and 2 pm. Therefore, the problem must be with the size of the classrooms and the classes: there are two many large classrooms (or conversely, too many small classes).

We conclude that 1) the university is not using all of its classroom space as efficiently as it might, and 2) the University could improve its classroom utilization by better balancing the size of its classrooms and the size of its classes. Our results do not guarantee that there are practical schedules that use the number of classrooms we determined because our process does not consider things like teachers' room preferences or class location requirements (English classes might not end up near the English department). We are more confident in the second conclusion because constraining the placement of classes further would only exacerbate the problem of having too many small classes.

4.3. Extensions

The University of Puget Sound is a small school, and its small number of classes limited us in an important respect. If we had had more classes to schedule we could have taken into account variables besides room size.

For instance, to incorporate a location constraint into our scheduler we could divide the classes into groups according to their size and then divide them according to their required location (English classes in the English building, for instance) and then applied QUICK_GROUP to these groups. When we build the bipartite graph with the resulting blocks we would only put edges between groups and classroom/time pairs that are the proper size and in the right building.

Using this technique of dividing classes into smaller and more carefully defined groups, we could incorporate a large number of different constraints. We run into a difficulty, however, when the collections of similar classes become too small. When this happens the number of left over classes increases dramatically, and the number of blocks needed to hold the classes increases correspondingly.

We originally attempted to incorporate building preference, but the blocks produced simply would not fit into the 51 classrooms available (mainly because the number of classrooms in some buildings are not sufficient to handle the classes for the departments residing in those buildings). One way around this would be to take the "left overs" and apply QUICK_GROUP to them. This would mean that some classes would end up in the wrong building (an occurrence not uncommon on college campuses and our campus in particular), but would still yield feasible schedules.

Acknowledgments

We thank the anonymous referees for pointing us to related literature and helping us clarify the experimental section.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network Flows, Prentice Hall, 1993.
2. M. Carter and C. Tovey, "When is the classroom assignment problem hard?" Operations Research, vol. 40, supp 1, pp. S28–S39, 1992.
3. X. Deng, H. Liu, J. Long, and B. Xiao, "Competitive analysis of network load balancing," Journal of Parallel and Distributed Computing, vol. 40, no. 2, pp. 162–172, 1997.
4. J. Ferland and S. Roy, "Timetabling problem for university as assignment of activities to resources," Computers and Operations Research, vol. 12, no. 2, pp. 207–218, 1985.
5. J. Ferland, S. Roy, and T. Loc, "The timetabling problem," in OR Models on Microcomputers, 1986, pp. 97–103.
6. P. Fizzano, "Centralized and distributed algorithms for network scheduling," PhD Thesis, Dartmouth College, 1995.
7. M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," Journal of the ACM, vol. 34, pp. 596–615, 1987.
8. D. Gale and L.S. Shapley, "College admissions and the stability of marraige," American Mathematical Monthly, vol. 69, pp. 9–14, 1962.
9. D. Gusfield and R.W. Irving, The Stable Marraige Problem: Structure and Algorithms, MIT Press, 1989.
10. D. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS, 1997.
11. B. Hoppe and E. Tardos, "The quickest transhipment problem," in Proceedings of the 6th Annual Symposium on Discrete Algorithms, 1995.
12. Max Planck Institute for Computer Science, "Leda, the library of efficient data structures and algorithms," v. 3.1, 1995.
13. S. Micali and V.V. Vazirani, "An $O(\sqrt{|v|} \cdot |E|)$ algorithm for finding maximum matching in general graphs," in 21st Annual Symposium on Foundations of Computer Science, 1980, pp. 17–27.