WHEN IS THE CLASSROOM ASSIGNMENT PROBLEM HARD?

MICHAEL W. CARTER
University of Toronto, Toronto, Ontario, Canada

CRAIG A. TOVEY
Georgia Institute of Technology, Atlanta, Georgia

(Received February 1989; revision received March 1990; accepted January 1991)

The classroom assignment (or hotel room or interval scheduling) problem is to assign classes, which meet at different time intervals, to rooms. Two classes may not meet simultaneously in the same room, nor may a class meet in two different rooms. Thousands of colleges and secondary schools face this problem every semester. There has been some confusion as to how hard this problem is. Many colleges claim that it is easy, while others complain that it is next to impossible. In the literature, some authors claim or conjecture polynomial time algorithms, while others develop heuristic approaches. The goal of this paper is to resolve the confusion by identifying cases where the problem will be easy and others where it will be hard. We focus on the kinds of cases that schedulers are apt to encounter in practice.

Hundreds of colleges and thousands of secondary schools across North America are faced with large classroom assignment problems every semester. The vast majority of institutions "solve" the problem manually, to a neverending chorus of complaints from instructors. A few places employ simple computer-assisted, first-come, first-served heuristics.

For example, the University of Waterloo, prior to 1987, used to develop a complete new assignment of classes to rooms every term. The manual process, typical of many other universities, utilized a large "room booking log" with one page for every room on campus. The Central Scheduling Office estimates that it took three experienced people two to three weeks to develop the initial room assignments.

By comparison, at many universities, the room assignments are essentially unchanged from one year to the next. When a course is discontinued, the room becomes free. When a new course is offered, they look for the best available room at that time. Consequently, people frequently must accept rooms that do not meet their requirements, and the problem is compounded annually.

There is some confusion as to the difficulty of classroom assignment. Many colleges and secondary schools claim that the problem is easy, while others complain that the task is next to impossible. In the literature, some authors claim or conjecture polynomial time algorithms, while others develop heuristic approaches. Our principal goal in this paper is to clear up some of this confusion. To do this, we give some new results, and also cite some work of others. It turns out that the difficulty depends not only on the objective and size of the problem, but on the underlying preferences of classes for rooms. By considering the kinds of situations actually encountered by schools, we can resolve much of the contradictory evidence that has been reported.

1. MATHEMATICAL FORMULATION OF THE PROBLEM

The usual formulation of the problem is expressed in terms of the variable x_{ij}, a 0–1 valued variable which is equal to 1 when class i is assigned to room j and is 0 otherwise. Time is divided into periods (e.g., every half hour, though all periods do not have to be the same length). We sometimes wish to fix the number of periods and refer to this case as the k-period problem. The constraints of this problem are

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \text{for } i = 1, 2, \ldots, n$$

and

$$\sum_{i \in P_k} x_{ij} \leq 1 \quad \text{for } j = 1, 2, \ldots, m; \quad \text{for all } k,$$

where P_k represents the set of all classes that meet in period k. The first constraint ensures that every class is assigned to some room, while the second constraint

Subject classifications: Education systems, operation: classroom assignment, course timetabling. Mathematics: computational complexity. Production/scheduling: applications.
Area of review: OPTIMIZATION.

0030-364X/92/4001-0S28 $01.25

prevents any room from having more than one class in the same period. We define three objectives for this problem:

Feasibility. Is there a feasible solution to the constraints above?

Satisfice. Given, for each class i, a set of rooms S_i that are *satisfactory*, is there a feasible solution to the constraints above that puts each class in a satisfactory room?

Optimize. Assume the existence of a linear cost model that adequately describes the problem and reflects reasonable preferences. This is a standard assumption and several papers describe such models (e.g., Glassey and Mizrach 1986, Carter 1989). The cost of assigning class i to room j is denoted c_{ij}. The objective is to minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}.$$

Obviously, the three objectives are listed in nondecreasing order of difficulty, regardless of other problem characteristics. This is because **Feasibility** can be modeled as a special case of **Satisfice,** which, in turn, is a special case of **Optimize.** In particular, if we assign a cost of 0 to all rooms j in the set S_i, and a cost of 1 elsewhere, then **Optimize** gives the **Satisfice** solution if the final total cost equals zero.

Of the three objectives, the one closest to problems in practice is probably **Satisfice.** Manual schedulers tend to think in terms of "acceptable" room assignments, (e.g., enough seats, not too far away, special facilities, etc.). They will not associate a specific numerical rating to every possible combination of assignments. Therefore, we will emphasize analysis of the **Satisfice** objective in what follows.

Implicit in our integer programming formulation is the constraint that a class is assigned to only one room. That is, students do not have to change rooms in the middle of a lecture! There are two versions of the problem depending on how the word *class* is interpreted. We call the first version the *1-day* or *interval* problem. Ordinarily, classes meet once in a day, from the beginning of one period to the beginning of some other period. In this case, the class requires the use of a room during an *interval*, hence, the alternative name "interval scheduling." But classes can meet more than once in a week. Some institutions require that each class must have *all* of its meetings in the same room. We call this version the *multiday* or *noninterval problem.* Normally, we will interpret the word class to mean a single meeting of a course, so hereafter in the classroom assignment problem, unless stated otherwise, we assume that each class requires the use of a room for some set of consecutive periods. For the sake of consistency, we will refer to these problems as *interval* or *noninterval.*

Before proceeding to the results, we remark that the classroom assignment model has several other applications, including hotel room assignments, shipyard scheduling, job scheduling on machines with varying capabilities, and assigning airport gates to flights (see Kolen, Lenstra and Papadimitriou 1987, Dondeti and Emmons 1986, Mangoubi and Mathaisal 1985). In the airport problem, **Optimize** is the appropriate objective; in the others **Satisfice** is applicable.

2. COMPLEXITY RESULTS

Observe that, for the interval problem, if there is no overlap of any class time periods, then even the **Optimize** problem is easily solvable by classical methods. For example, if all classes are one hour in length and always begin on the hour, then each hour of the day can be solved as a separate assignment problem.

Observation 1. The 1-period classroom assignment **Optimize** problem is polynomially solvable.

In many practical problems, classes will take on a variety of different sizes. Secondary schools often utilize single and double period courses. At the university level, classes can be 1, 1.5, 2, 3 or 4 hours in length, beginning every half hour.

Several heuristics for solving this problem have appeared in the literature (Mulvey 1982, Ferland and Roy 1985, Glassey and Mizrach 1986, Gosselin and Truchon 1986, and Carter 1989). Two of these papers (Glassey and Mizrach 1986, Gosselin and Truchon 1986) imply that it might be possible to develop a polynomial algorithm. Papers by Arkin and Silverberg (1987) and Kolen, Lenstra and Papadimitriou prove that the general problem is NP-complete, and we are interested resolving this contradiction.

Gosselin and Truchon solve each day separately and model the problem as a linear program. To keep the number of variables and constraints within reasonable limits, they aggregate the rooms into groups of similar size and facilities and aggregate classroom requests by size, facilities and time period. They discovered that the corresponding linear program always produced integer solutions, although they were unable to prove

or disprove this statement in general. They tested their procedure on several test problems and on actual data from Université Laval in Quebec.

In an appendix, Glassey and Mizrach claimed that the four-period problem, when classes are either one or two periods long, can be transformed into a network problem with a polynomial solution. The proof of this result was not included in the paper because the problem addressed was more complex.

These results lead us to consider the complexity of the problem. When is it NP-complete and under what conditions does a polynomial solution exist? Our first theorem concerns the most general formulation of the constraints.

Theorem 1. *The noninterval classroom assignment* **Feasibility** *problem is NP-complete. We phrase the problem in the form: Does there exist a feasible assignment of classes to rooms? We assume that classes may meet several times per week, but they must be assigned to the same room for all meetings. (Note that this assumption was used by Glassey and Mizrach at Berkeley.)*

Proof. Clearly, classroom assignment belongs to NP. That is, given any assignment, we can test feasibility in polynomial time. We will prove that the graph K-colorability problem is polynomially reducible to the classroom assignment problem. Since K-colorability is NP-complete (Garey and Johnson 1979), therefore, classroom assignment is NP-complete.

Suppose that we are given a general graph coloring problem, i.e., can the vertices of a given graph be colored using at most K colors, such that no two adjacent vertices have the same color? Let n be the number of vertices in the graph and p be the number of edges. We construct an equivalent classroom assignment problem with K-identical classrooms, n classes and p time periods. If edge e_k connects vertices i and j, then classes i and j must both meet in period k. This classroom assignment problem has a feasible solution if and only if the corresponding graph is K-colorable. Therefore, the noninterval classroom assignment problem is NP-complete, as required.

So, for the noninterval case of classroom assignment, even the **Feasibility** question is NP-complete. Since graph K-colorability is hard for $K = 3$, the noninterval feasibility problem is hard even for only three (identical) rooms. This problem does not need to be very complex in order to be *hard*. Consider the following simple example based on coloring the nodes of a cycle of length five with two colors.

Example 1

Course	Periods
A	1, 2
B	2, 3
C	3, 4
D	4, 5
E	1, 5

Given two equal rooms, the associated linear program (LP) admits a feasible solution with every class in half of each room. But, of course, one can easily recognize that this problem has no feasible solution, even though E is the only "noninterval" course. However, if we were to bury several such odd cycles within a practical problem, it would become very difficult to verify the nonexistence of a feasible solution.

The noninterval problem states that, when a class meets more than once a week, every meeting must occur in the same room. This restriction is often desirable, but it is not usually considered absolutely necessary. In particular, when the problem is solved one day at a time, as in Carter, and Gosselin and Truchon, each class meets only once and the coloring reduction no longer applies, i.e., each meeting is effectively treated as a separate class and the whole week becomes an interval problem. In machine scheduling applications, frequently an operation may not be interrupted (so it must be performed on one machine), but it is not necessary that all the operations for a job he performed on the same machine. In this case, the **Feasibility** question becomes quite easy.

Observation 2. The interval classroom assignment **Feasibility** problem is polynomially solvable in $O(n)$ time.

Proof. From the proof of Theorem 1, clearly it is sufficient to color the appropriate graph. But this graph is an interval graph, which is well known to be easily colorable. (It is perfect, so the minimum number of rooms needed is equal to the maximum number of classes in session at any time.) (See Golombic 1980, chapter 8.)

Let us now consider the **Satisfice** case. Here we have a situation actually faced by many schools.

Theorem 2. *The interval classroom assignment* **Satisfice** *problem is NP-complete.*

This result, based on a reduction from 3-SAT, was published independently by Arkin and Silverberg in the context of job scheduling, and we will omit the proof. However, we can strengthen their result considerably with the use of an intermediate reduction of

3-SAT. Theorem 3 represents a minor variation of Theorem 2.1 in Tovey (1984) (also hinted at in Garey and Johnson without proof).

Theorem 3. *Any instance of* 3-*SAT can be reduced to an instance of SAT such that*:

1. Each clause has two or three literals. Call these 2-clauses and 3-clauses, respectively.
2. No two 3-clauses have any literal in common (where X_i and $\bar{X}_i$ are different literals).
3. No two 2-clauses have any literal in common.

Proof. Given any instance of 3-SAT and consider any variable X, which has a duplicated literal. Suppose that the variable X appears in r clauses as X and in s clauses as $\bar{X}$. Let $k = \max\{r, s\}$. Create k new variables $Y_1, \ldots, Y_k$, and replace the ith occurrence of X with Y_i for each $i = 1, \ldots, r$ (and $\bar{X}$ with $\bar{Y}_i$ for each $i = 1, \ldots, s$). Append the clause $\{Y_i \vee \bar{Y}_{i+1}\}$ for $i = 1, \ldots, k-1$ and the clause $\{Y_k \vee \bar{Y}_1\}$.

In this new instance, the clause $\{Y_i \vee \bar{Y}_{i+1}\}$ implies that, if Y_i is false, then Y_{i+1} must also be false. The cyclic structure of the clauses therefore forces the Y_i variables to be either all true or all false. Moreover, the transformation requires polynomial time. It is easy to verify that, in the new instance, every literal occurs at most once in all of the 2-clauses and at most once in all of the 3-clauses, as required.

We can now show the stronger result.

Theorem 4. *The interval classroom assignment* **Satisfice** *problem is NP-complete even when the problem is restricted to two periods.*

Proof. Consider any special instance of SAT as defined in Theorem 3. Suppose that there are n variables X_i. Let each variable represent two rooms called TRUE_i and FALSE_i for a total of $2n$ rooms. For each 2-clause, define a class C_j^2 which occurs in period 1. Each of them has two preferred rooms corresponding to the two literals in the clause. For example, if 2-clause j contains the literal $\{X_1 \vee \bar{X}_2\}$, then class C_j^2 prefers rooms TRUE_1 and FALSE_2. Similarly, for each 3-clause, define a class C_j^3 occurring only in period 2 and having three preferred rooms. Observe that, since none of the 2-clauses have a common literal, all of the classes in period 1 prefer different rooms. Similarly, none of the period-2 classes have any preferred rooms in common.

We now define a set of n classes, L_i that meet for both periods and prefer rooms TRUE_i and FALSE_i. Observe that, if class L_i gets preferred room TRUE_i, then all other classes C_j^2 and C_j^3 can only use room FALSE_i, i.e., the purpose of the L_i is simply to force each variable to be always true (if L_i uses room FALSE_i) or always false (if L_i uses room TRUE_i). Clearly, all classes can be assigned to their preferred rooms if and only if the corresponding 3-SAT instance is satisfiable.

Theorem 4 states that the classroom assignment problem has the potential to become very nasty. However, it does not help us to understand whether or not difficult problems are likely to occur in a practical setting. The following example illustrates that the problem does not have to be very complex in order to be computationally difficult.

Example 2

Class	Periods	Satisfactory Rooms
A	1	1, 2
B	1, 2	2, 3
C	1	3, 4
D	1, 2	1, 4
E	2	1, 3
F	2	2, 4

It is easy to show that we cannot assign all classes to satisfactory rooms. For example, in period 1, if classes A, B, C and D are assigned to rooms 1, 2, 3 and 4, respectively, then there is no available satisfactory room for class F in period 2, i.e., classes B and D use both rooms 2 and 4. Conversely, if classes A, B, C and D are assigned to rooms 2, 3, 4 and 1, respectively, the only other solution, then there is no satisfactory room for class E. In both cases, at least one class must go in a room that is not satisfactory.

However, it is also easy to show that the optimal LP relaxation finds a solution with $x_{ij} = 0.5$ for each class i in each of its two preferred rooms. The LP finds an answer when, in fact, no feasible solution exists.

After we had discovered the preceding example, we realized that it could be derived from Theorems 3 and 4 in the following way. Consider the simplest unsatisfiable SAT problem: $(X) \wedge (\bar{X})$. Convert it to: $(X \vee X) \wedge (\bar{X} \vee \bar{X})$, the simplest unsatisfiable instance whose LP relaxation has a feasible solution. That is,

$$X + X \geq 1$$

$$\bar{X} + \bar{X} \geq 1.$$

Set $X = \bar{X} = 0.5$.

Now, apply the transformation of Theorem 3 to get the SAT instance:

$(Y_1 \vee Y_2) \wedge (\bar{Y}_1 \vee \bar{Y}_2) \wedge (Y_1 \vee \bar{Y}_2) \wedge (Y_2 \vee \bar{Y}_1)$.

Finally, apply the transformation of Theorem 4 treating the first two clauses as 3-clauses (corresponding to classes E and F in period 2). This yields Example 2 where:

Room 1 = TRUE_1
Room 2 = FALSE_2
Room 3 = TRUE_2
Room 4 = FALSE_1.

This derivation illustrates an interesting use of an NP-completeness proof. The proof not only tells us that the problem is hard, but also *what kind of instances* of the problem are hard. This could be a useful, general tool for finding difficult instances of NP-complete problems.

Arkin and Silverberg have developed a polynomial algorithm for the interval **Satisfice** problem when the number of rooms is fixed at some value m. Their dynamic programming approach requires $O(n^{m+1})$ operations, where n is the number of classes. Clearly, this approach is reasonable only when the number of rooms is very small. Their construction can also be modified to solve the **Optimize** problem.

To summarize, even **Feasibility** is hard for noninterval classroom assignment. **Feasibility** is easy for interval classroom assignment, but **Satisfice** is hard, once there are two or more periods in the day and more than two types of rooms. If there is only one period, even **Optimize** is easily solved as an assignment problem.

3. MONOTONIC PREFERENCES, OR "I WANT A BIGGER ROOM!"

The preceding results lead to some interesting questions. If the problem is NP-complete for two periods, then how did Glassey and Mizrach get a polynomial algorithm for four periods, and why do Gosselin and Truchon get integer solutions to the LP?

A partial explanation can be found by looking at the assumptions underlying the construction of the 3-SAT model. There are no restrictions on the patterns of room preferences. In particular, in Example 2 of Section 2, every pair of rooms is preferred by one of the classes. Our model of the problem assumes that there exists some subset of the available rooms that can be preferred in almost any combination.

In the problem described by Carter, rooms may be considered desirable for a variety of different reasons. Location is one criterion, but nonadjacent buildings may be preferred. Some people prefer windowless rooms for projection purposes, while others prefer lots of sunlight. Some people request special facilities, such as cinematography, televisions, computer hook-up capabilities, and large blackboards. Certain seating types are more appropriate for some types of classes. In the model described, instructors can simply specify a list of preferred rooms. One can visualize a wide variety of combinations across 2,400 classes.

By contrast, the model described by Gosselin and Truchon allows for each class to be assigned a list of possible rooms in the order of preference. However, in their testing, they used a single preference order for all the rooms, primarily based on space. Each class preference was based on the sublist, beginning with the first room large enough to accommodate the class. For these problems, the LP always found integer solutions.

We will define this special case of the **Satisfice** problem as having a monotonic preference function. Specifically, there exists an ordering of the rooms, R_1, $R_2, \ldots, R_m$, such that if a class i will accept room R_j, then class i will also accept any room R_k, where $k > j$.

Kolen, Lenstra and Papadimitriou considered a related machine scheduling problem. We are given a set of jobs with fixed start times and durations and a set of machines with different processor capacities, where each job has a minimum required processor size. We must determine whether or not there exists a feasible assignment of jobs to machines. This is equivalent to the interval monotonic **Satisfice** problem, i.e., where the only assignment constraint is that the room has enough seats to accommodate the class.

Kolen, Lenstra and Papadimitriou, and Dondeti and Emmons were able to show the existence of a polynomial algorithm when there are only two sizes of processors (two room categories). This result is of major importance for many secondary schools. Classrooms are often designed with a fixed capacity and any class will fit in any room (with possibly one or two large auditoriums available for combined classes). These problems are essentially trivial even when classes are of many different lengths and all rooms are not always available as long as there are enough rooms in each time period. The problem remains easy when special purpose space such as chemistry labs are included, provided that such space can only be used by chemistry classes, i.e., the problem can be decomposed by the type of space required.

Kolen, Lenstra and Papadimitriou also showed that the interval classroom assignment **Satisfice** problem

with monotonic preferences is NP-hard if there are three or more room categories. They employ a reduction from "numerical 3-dimensional matching" (Garey and Johnson). The construction requires a fairly large number of periods proportional to the size of the numbers in the corresponding matching problem. We have been able to construct an example to illustrate the type of situation that can occur.

Figure 1 contains a set of 28 classes that must be assigned to eight rooms over eleven periods. There are two large rooms, two medium rooms and four small rooms available. The classes labeled I require a large room, while those labeled II are medium sized classes that need a medium or large room. The remaining unlabeled classes can go in any room.

It is easy to verify that this problem has no feasible solution. However, if we relax the integrality restriction and solve it as a linear program, we discover that there is a feasible fractional solution. Specifically, classes marked (1) have half a medium and half a small room; classes marked (2) have half a large and half a small room; and classes marked (3) have half a large and half a medium room. All other classes are assigned to a single room of the required size.

Although we are not certain that this represents a *smallest possible* example, we believe that it is very close. It appears unlikely that one could construct a situation with a gap between the LP solution and the integer optima with much fewer than eleven periods.

This helps to explain the results of Gosselin and Truchon. With a monotonic preference function and fewer than eleven periods, we would expect that the LP would always find an integer solution. It is also possible that Glassey and Mizrach used a monotonic assumption when they apparently proved the existence of a polynomial solution for the 4-period problem.

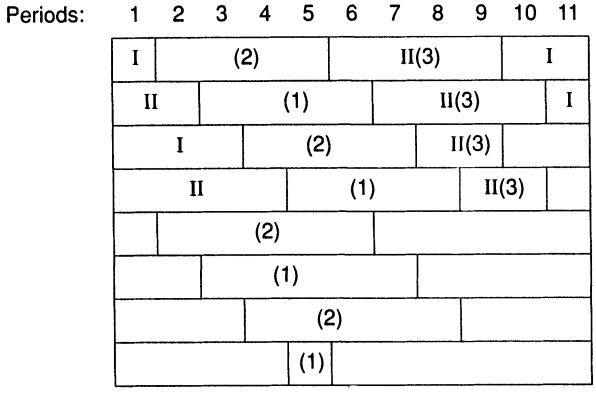

Figure 1. Example 3.

This "11-period" conjecture may have important practical implications. If each day is solved separately and the periods are all multiples of an hour, then the normal day will be at most ten periods long. This is true for many schools. Gosselin and Truchon describe the problem at Université Laval as having this structure, for example.

At the University of Waterloo, Carter describes a problem that uses half-hour periods. However, each half day is solved separately, so again, no subproblem has more than ten periods. The conjecture does not help in this case though because that problem is not monotonic.

Our next result shows that, if the number of periods is fixed at any level, then there is a polynomial solution to the problem, even for the noninterval case.

Theorem 5. *For any fixed k, the k-period noninterval (respectively, interval) monotonic* **Satisfice** *problem can be solved in polynomial time* $O(n^{2^k}m)$ *(respectively,* $O(n^{k^2/2}m)$*).*

Proof. We use dynamic programming. Let $K = 2^k - 1$. There are K different possible *types* of schedules for each class, depending on whether the class meets or not during period i: $i = 1, \ldots, k$

(For the interval problem

$$K \text{ can be taken as } \binom{k+1}{2}.)$$

Arbitrarily index these K types by $j = 1, \ldots, K$. For example, if $k = 2$, the types could be 1, 2 and 1–2 for $j = 1$, 2 and 3, respectively. We let **F** (for **Feasible**) denote the collection of all 0–1 K-vectors representing nonoverlapping sets of class types. For example, with $k = 2$ and the indexing above,

$$F = \{(0, 0, 0); (1, 0, 0); (0, 1, 0); (0, 0, 1); (1, 1, 0)\}$$

since type 3 overlaps with types 1 and 2. Each member $f \in F$ represents a feasible combination of classes that can be assigned to a single room.

For ease of intuitive understanding, we phrase preferences in terms of room size. The smallest room is the least preferred. Arrange the m rooms from smallest to largest. Let C_i denote the K-vector of the number of classes of each type which would fit into room i but not into room $i - 1$. Thus, C_1 = the number of classes of each type that fit into room 1.

First, we give a verbal description of the algorithm. Our stages are to consider each room, in turn, assigning classes to it. When we finish assigning classes to a room, our state is characterized by how many of each

type of class have not yet been assigned a room, but are small enough to fit into the next room under consideration. A state, therefore, can be characterized by a K-vector s of integers, where s_j equals the number of classes of type j ($j = 1, \ldots, K$). The key idea is this: Any two classes of the same type that are small enough to fit into the room under consideration are indistinguishable. Since we consider the rooms in order of increasing size, all we need to keep track of is this K-vector. We do not need to keep track of the exact size requirements of the unassigned classes, because they are guaranteed to be small enough to fit into the other rooms. As we start stage i, S_i is a collection of K-vectors that are the possible configurations of classes that are unassigned, and which could fit into room i, that is, S_i is the set of possible state vectors s at stage i. The process terminates affirmatively if and only if S_{m+1} contains the 0 vector, i.e., after using the biggest room, all classes have been assigned.

A more precise description of the algorithm follows:

Inputs: K; m; C_i: $i = 1, \ldots, m$; F.

Step 1. **Initialize:** Set $i \leftarrow 1$; $S_1 \leftarrow \{C_1\}$.

Step 2. **Put Classes in Room *i*:** For each $s \in S_i$ set $R(s) \leftarrow \{(s - f) |$ for all $f \in F$; $(s - f) \geqslant 0\}$, i.e., $R(s)$ represents the set of all configurations of *remaining* unassigned classes for each feasible assignment, $f \leqslant s$.

Step 3. **Consolidate:** Set $R \leftarrow \cup_{s \in S_i} R(s)$. R is the set of all possible configurations of unassigned classes of size i or less.

Step 4. **Add Classes:** Set $S_{i+1} \leftarrow R + C_{i+1} \equiv \{(\mathrm{r} + \mathrm{C}_{i+1}) | \mathrm{r} \in \mathrm{R}\}$.

Step 5. **Next Stage:** Set $i \leftarrow i + 1$; if $i \leqslant m$ go to Step 2.

Step 6. **End:** If $0 \in S_{m+1}$ return YES, otherwise return NO.

It is obvious that if the algorithm returns YES, there is a satisfactory solution. Conversely, if T were a satisficing solution, it would induce a sequence of s^i, where $s^{m+1} = 0$, $s^m =$ the classes T assigns to the biggest rooms, etc. Then it is easy to see inductively that if $s^i \in S_i$, then $s^{i+1} \in S_{i+1}$, and so the algorithm will return YES. Therefore, the algorithm works correctly.

Now we verify that the algorithm runs in polynomial time. There are m iterations, one per room. During any iteration, how many K-vectors can S have? If there are n classes, an upper bound would be the number of ways to distribute n or fewer identical balls into $K + 1$ nonidentical runs. This is just

$$\binom{n + K}{K}$$

(e.g., see Lovasz 1979, p. 14), which is dominated by n^K, a function polynomial (though huge) in n. Since $|F| \leqslant 2^K$, Step 2 takes at most

$$\binom{n + K}{K} 2^K \leqslant (2n)^K$$

time. Clearly Step 2 dominates, so the total time is $O((2n)^K m) = O(n^K m)$, a polynomial as claimed.

This dynamic programming approach can be streamlined considerably to improve its efficiency. In Step 3, we construct the set of all possible K-vectors of unassigned classes. Clearly, only the minimal elements of this set need to be considered; the others are dominated. For example, if two options A and B are identical, except that option A leaves 20 classes meeting from 8:00 to 10:00 unassigned, and option B leaves 19 of this type unassigned, obviously B dominates A.

There is another more subtle kind of dominance. We develop it by example. Suppose that the smallest room holds ten students, and there is only one room this size. Suppose that there are three classes, denoted 1, 2, 3 with $\leqslant 10$ students, with meeting times 8:00–9:00, 9:00–10:00, and 8:00–10:00, respectively. In the first iteration of the dynamic programming algorithm, there are five possible options for assigning classes. The obvious dominance from the previous paragraph reduces this to two options, namely, {1, 2} and {3}. We can further reduce this to one option, namely {3}. This option dominates {1, 2} because any room that would accommodate {3} would accommodate {1, 2}. (This is where we use the monotonicity that all remaining rooms are big enough.) But a subsequent classroom assignment that accommodated 1 and 2 might not accommodate 3, because 1 and 2 might be in different rooms. Essentially, leaving {1, 2} unassigned has more flexibility than leaving {3}.

These dominance ideas can greatly improve the theoretical efficiency of the dynamic programming algorithm outlined above. For instance, for $k = 3$ periods, the brute-force algorithm is $O(n^7 m)$, while the streamlined algorithm is $O(n^3 m)$.

Two other ways to speed up the algorithm are:

- K can be reduced to equal the number of different types of classes that exist (recall that K was the maximum possible such number). For example, as we pointed out above, for the interval problem, we can take

$$K = \binom{k + 1}{2}.$$

- If there are two or more rooms of the same size, Steps 2–4 can be performed simultaneously for these rooms. This reduces the m in the time bound to the number of sizes that rooms come in, which is typically smaller than the number of rooms.

We conjecture that, for the interval problem, one could construct an algorithm with a bound of $O(n^k m)$, and perhaps not much larger for noninterval problems.

This approach may be of practical significance. Many schools will divide the day into morning and afternoon subproblems. This produces two separable problems per day with four or five periods each and dynamic programming may be a viable alternative. In short, the problem is easy (i.e., polynomial) when either the number of periods or the number of rooms is very small.

The results of Theorem 5 can easily be extended to the noninterval **Feasibility** problem with arbitrary preference constraints.

Corollary. *Feasibility of the k-period noninterval problem can be solved in polynomial time.*

Proof. This can be modeled as the k-period **Satisfice** problem, where all rooms are assumed to be satisfactory for all classes, i.e., **Feasibility** ignores preferences. Hence, by Theorem 5, it has a polynomial solution.

4. HOW AND WHEN TO BE GREEDY

In the majority of practical situations, room assignment is performed manually using a one-pass "greedy" procedure even when the problem is noninterval. The large classes will be assigned to the largest available rooms. The primary objective is to find a feasible solution based on classroom sizes. The initial feasible solution will later be improved using simple exchanges to try to satisfy some of the secondary requirements, but feasiblility with respect to size dominates the first pass. Therefore, we are interested in analyzing the effectiveness of the greedy procedure. We want to explore the conditions under which the greedy routine would be optimal.

Suppose that we are given a set of n classes and a set of m rooms. Assume that the rooms have been ordered by the number of seats and that any class can be assigned to a room if the capacity of the room is large enough to accommodate the class. The following *greedy* algorithm will find a feasible solution to the problem if one exists when the number of periods is at most three.

Three-Period Algorithm

Add a sufficient number of small (unrestricted) one-period dummy classes so that the total demand for rooms equals the supply. Fill the smallest room first. If possible, use a 3-period class. Otherwise, look for a feasible combination of a 2-period and a 1-period class. Finally, look for three 1-period classes. Assign the chosen block to the smallest room. Remove the room and the class(es) from the list and repeat until all classes are assigned. At any stage of the algorithm, if a room has no feasible class assignment, the routine terminates.

Theorem 6. *The Three-Period Algorithm will always find a feasible solution if one exists.*

Proof. Assume that there exists a feasible solution, S^*. We will show that we can always modify S^*, such that it agrees with each iteration of the algorithm and retains the feasiblility property.

Consider the smallest room in S^*. Suppose that the algorithm would have selected a 3-period class W for this room, but the schedule S^* has a block of classes B. Room j is the smallest. Observe that we can always interchange class W with block B.

Period	1	2	3
Room i	W		
Room j	B		

Now, suppose that the algorithm would have selected a block with a 2-period class. Assume that the

Period	1	2	3
Room i	X2		
Room j	C		X1
Room k	B		

rooms are ordered from largest to smallest, not necessarily consecutive, and that the algorithm would have chosen the block {X2, X1}. Since class X2 is feasible in room k, it must also be feasible in room j. Therefore, if we interchange block C with class X2, the solution will retain feasibility. Now, we can interchange the block {X2, X1} with the block B. The other possible cases are similar.

Finally, suppose that the algorithm would have chosen three 1-period classes: X, Y and Z. The

schedule S^* takes the following form:

Period	1	2	3
Room h		Y	
Room i	X	D	
Room j	C		Z
Room k	B		

By the definition of the algorithm, there are no feasible combinations of 2-period or 3-period classes in the smallest room. Therefore, block B must consist of three 1-period classes, and clearly, we can interchange each of these classes with X, Y and Z, respectively.

In each case, after performing the interchange, remove the smallest room from S^* and repeat the construction. The resulting assignment must be feasible. Hence, the algorithm will only fail when no feasible solution exists.

This algorithm will also work when applied to monotone interval problems with any number of periods p under a special overlap restriction. We define a *partial overlap* between a pair of classes to mean that the classes have at least one period in common and *each* class has at least one period not in common with the other, e.g., class A in periods 1, 2 and 3 has a partial overlap with class B in periods 3 and 4, but *not* with class C in periods 2 and 3. It turns out that *nested* class times (if two intervals intersect, one completely contains the other), are generally easier than partial overlap times. A common example of *nested* structure in practice can be seen in many secondary schools that only use single and double period classes (where the doubles do not overlap).

Nested Class Times Interval Monotone Satisfice Greedy Algorithm

As before, we initialize by adding enough single period dummy classes to fill all rooms in each of the p periods. Fill the smallest room first. If possible, use a p-period class. Otherwise, look for a feasible combination of a $(p-1)$-period class with a 1-period class. Finally, look for the longest class that will fit in the smallest room and assign it breaking ties arbitrarily. Fill any remaining time in the smallest room iteratively with the longest class that will fit in the room. Repeat until the room is full.

Remove the room and the assigned classes from the problem and continue with the next smallest room. At any stage of the algorithm, if a room cannot be filled completely, then there is no feasible solution.

Theorem 7. *The Greedy Algorithm will always find a feasible solution when one exists provided that no pair of classes of length from 2 to p-2 periods has a partial overlap.*

For example, for a 5-period problem, we do not allow any 2- and 3-period classes to have partial overlap. However, we do allow 4-period classes to have partial overlap with 2, 3 or 4-period classes.

Proof. As before, suppose that there exists a feasible assignment of classes to rooms, S^*. We wish to show that we can always modify the schedule S^*, such that it agrees with an arbitrary ordering of the classes, as selected by the algorithm.

Consider the smallest room in S^*. Suppose that the algorithm would have selected a p-period class W for this room, but the schedule S^* has a block of classes B.

Period	1 ... p
Room i	W
Room j	B

Room j is the smallest. Observe that we can always interchange class W with block B.

Now, suppose that the algorithm would have selected a block with a $(p-1)$-period class.

Period	1 ... p	
Room i	X2	
Room j	C	X1
Room k	B	

Assume that the rooms are ordered from largest to smallest, not necessarily consecutive, and that the algorithm would have chosen the block {X2, X1}. Since class X2 is feasible in room k, it must also be feasible in room j. Therefore, if we interchange block C with class X2, the solution will retain feasibility. Now, we can interchange the block {X2, X1} with the block B. The other possible cases are similar.

Finally, suppose that the algorithm would have selected some q-period class X, where $2 \leq q \leq p-2$. The schedule S^* contains a block of classes B. By definition of X, there are no feasible classes of length greater than q, and we assumed that there is no partial overlap. Therefore, we conclude that the block B *must* contain a partition, as shown below.

Period	1 ... p		
Room i		X	
Room j	B1	B2	B3

We can always interchange X with the block of classes B2 and retain feasibility.

Moreover, the definition of X implies that *every* possible class that can be assigned to the smallest room must be completely contained in the periods defined by the blocks B1, B2 and B3. Hence, we can consider the remaining subproblems, B1 and B3 separately, i.e., the algorithm selects the longest class that will fit in block B1 (or B3), which may again subdivide the block.

This process continues until the longest class is a single period. At that point, *all* feasible candidates for the smallest room must be 1-period classes, and we can perform direct interchanges.

It is interesting and somewhat surprising that this algorithm does *not* work in reverse order! If we put the largest classes into the largest available rooms, we can create a situation where the algorithm fails to find a feasible solution when one exists. Consider the following simple example.

Example 4

Class	Size	Periods
A	75	1, 2
B	90	3
C	80	1
D	75	2, 3
E	70	1
F	70	2
G	70	3

Suppose that we are given three rooms i, j and k of sizes 90, 80 and 70, respectively. It is easy to verify that a feasible solution results from assigning A and B to room i, C and D to room j, and E, F and G to room k. However, if we assign the large classes first to the largest room, we will put classes B and C in room i for periods 1 and 3. Class F is the only class that will now fit in period 2, so we assign it to room i. Clearly, class A must be matched with G, and class D must be matched with E. But both of these combinations require 75 seats, and hence, there is no feasible completion. Classes A and B cannot be assigned to the same room in any feasible schedule.

This is particularly significant since, as we mentioned earlier, that is precisely the rule which most schedulers use! They put the large classes in the large rooms. Based on our results, it would be better to work in the reverse order and assign all the small classes to the smallest rooms first.

SUMMARY

We summarize the results we have discussed in Table I. As we stated in the Introduction, both the nature of class times and the types of preferences strongly affect the difficulty of the problem. Horizontally, the classifications range along the different kinds of preferences, from monotonic with two room sizes to arbitrary preferences. Problems get harder as we move from left to right. Vertically, the table runs through different kinds of course times from the 1-period problem to the noninterval problems. Things tend to get more difficult as we move from top to bottom except that interval does not dominate k-period noninterval. That is, **Feasibility** is easier for "interval," but **Satisfice** is harder.

In the table, we use the abbreviations "Feas" and

Table I
Summary of Complexity Results

	Monotonic 2-Room Types	Monotonic 3-Room Types	Monotonic	Arbitrary
1-Period	Optimization is easily solved $O(n^3)$ assignment			
2-Period	Satis Easy	Satis Easy	Satis Easy (Greedy)	Feas Easy Satis NPC
k-Period Interval	Satis Easy (Matching)	Satis Poly $O(n^{k^2/2})$	Satis Poly $O(n^{k^2/2})$	Feas Easy Satis NPC
k-Period Noninterval	Satis Poly $O(n^{2^k})$	Satis Poly $O(n^{2^k})$	Satis Poly $O(n^{2^k})$	Feas Poly Satis NPC
Interval	Feas Easy Satis Easy (Matching)	Feas Easy Satis NPC	Feas Easy Satis NPC	Feas Easy Satis NPC
Noninterval	Feasibility problem is NP-complete			

"Satis" for the problems **Feasibility** and **Satisfice,** respectively. We say that a problem is "Easy" if there is a standard, low-order polynomial algorithm. A problem is "Poly" for polynomial if there exists a possibly higher order or nonstandard method. A problem is "NPC" if it belongs to the class of NP-complete problems. The table does not include the results in Section 5 on the interval, nonoverlap monotone problem, or the Arkin and Silverberg result for a fixed number of rooms, but it does cover just about everything else. In particular, we have completely classified all of the **Satisfice** instances, the most practical problem.

6. CONCLUSIONS

Classroom assignment is deceptively difficult in that the problems *look* easy. We first proved that **Feasibility** is NP-complete when we insist on assigning the same room to each class that meets on more than one day. Furthermore, we have shown that the interval **Satisfice** problem is NP-complete even when there are only two periods. When the problem is further restricted to a monotonic preference function, it remains NP-complete (for an unlimited number of periods), but it may require as many as eleven periods to construct nonintegral solutions to the corresponding linear program.

On the positive side, we have given a polynomial algorithm for the noninterval problem when the number of periods is fixed under a monotonic preference function. This may be useful because the number of periods in practical problems is often very small. We also showed that a greedy algorithm can solve a restricted version of the problem with no partial overlap allowed. This helps to explain why many people view the problem as relatively simple.

Also on the positive side, we believe that we have cleared up several misconceptions about the general class of problems. We think that the summary section puts most of the claims and theorems in the literature into perspective.

A remark on the use of complexity theory: We would be remiss if we did not mention some effective heuristics and integer programming techniques for these problems (Carter 1989, Ferland and Roy 1985, Glassey and Mizrach 1986, Gosselin and Truchon 1986, Mangoubi and Mathaisal 1985, Mulvey 1982). We do not suggest complexity as an excuse for laziness . . . , "If it is NP-complete, don't try to solve it." Rather, complexity theory gives us an idea of what kind of techniques to use. As we have seen, it can also help us to identify difficult cases (that cannot be solved using an LP).

We have assumed that the monotonic interval **Satisfice** problem is a reasonable model for the approach used by most people in practice. We suspect that a more appropriate model would employ a *single-peaked* preference function. Under this model, each class would have its own preferred room (e.g., the right size). Preferences decrease as the room sizes increase *or* decrease away from the ideal size (cf, Fishburn 1973). Single-peaked functions include monotonic preferences where every class prefers the largest room. Hence, they would occur between "monotone" and "arbitrary" in Table I. Clearly, the interval **Satisfice** problem would remain NP-complete. However, it is not immediately clear how this structure would affect the 2-period or the k-period problems. We leave this as a topic for future study.

ACKNOWLEDGMENT

The authors would like to extend their thanks to the two anonymous referees who read the paper carefully and made several suggestions for improvements. This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (OGP0001359), by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032), and by a matching grant from Digital Equipment Corporation.

REFERENCES

ARKIN, E. M., AND E. B. SILVERBERG. 1987. Scheduling Jobs With Fixed Start and End Times. *Discrete Appl. Math.* **18,** 1–8.

CARTER, M. W. 1989. A Lagrangian Relaxation Approach to the Classroom Assignment Problem. *INFOR* **27,** 230–246.

DONDETI, V. R., AND H. EMMONS. 1986. Resource Requirements for Scheduling With Different Processor Sizes—Part I. Technical Memorandum 577, Department of Operations Research, Case Western Reserve University, Cleveland, Ohio.

FERLAND, J. A., AND S. ROY. 1985. Timetabling Problem for University as Assignment of Activities to Resources. *Comput. Opns. Res.* **12,** 207–218.

FISHBURN, P. C. 1973. *The Theory of Social Choice.* Princeton University Press, Princeton, N.J.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco.

GLASSEY, C. R., AND M. MIZRACH. 1986. A Decision Support System for Assigning Classes to Rooms. *Interfaces* **16**(5), 92–100.

GOLUMBIC, M. C. 1980. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York.

GOSSELIN, K., AND M. TRUCHON. 1986. Allocation of Classrooms by Linear Programming. *J. Opns. Res. Soc.* **37,** 561–569.

KOLEN, A., J. K. LENSTRA, AND C. PAPADIMITRIOU. 1987. Interval Scheduling Problems. Unpublished Working Paper, Centre of Mathematics and Computer Science, C.W.I., Amststerdam.

LOVASZ, L. 1979. *Combinatorial Problems and Exercises.* North-Holland, Amsterdam.

MANGOUBI, R. S., AND D. F. X. MATHAISAL. 1985. Optimizing Gate Assignments at Airport Terminals. *Trans. Sci.* **19,** 173–188.

MULVEY, J. M. 1982. A Classroom/Time Assignment Model. *Eur. J. Opnl. Res.* **9,** 64–70.

TOVEY, C. A. 1984. A Simplified NP-Complete Satisfiability Problem. *Discrete Appl. Math.* **8,** 85–89.