The Complexity of Clerkship Scheduling

Jonathan Feldman
Computer Science Honors Thesis
Dartmouth College
June, 1997

Professor Clifford Stein, Advisor
Department of Computer Science
Dartmouth College

Abstract

Medical students must complete a clerkship program in their fourth year. Individual students have preferences for the clerkships to which they are assigned. However, individual hospitals also have capacities on how many students may be assigned to each clerkship. The problem of scheduling medical students to clerkships is formalized. The problem is then placed in a theoretical framework, and the most general case of Clerkship Scheduling is proven NP-hard. A detailed approximation algorithm is given, and an implementation of this algorithm is discussed and tested.

PCS-TR97-316

Introduction

In the fourth year of medical school, every student must complete a clerkship program in several different areas of medicine. Each clerkship is for a different specialty of medicine, and each is offered at different times and locations. Additionally, hospitals can only accommodate a certain number of students for each clerkship. Students select their choices of assignments; some are granted, some are not. Ideally, the algorithm to schedule these students would maximize their satisfaction, and still obey the capacities that the hospitals set forth.

In the past, Dartmouth Medical School (DMS) has used a system where students are assigned random numbers, and given preference based on those numbers. When a student's number came up, they selected a schedule based on what was available. Students were also allowed to trade with each other afterwards.

DMS is currently in the process of automating their system. Each student will be allowed to quantify his or her satisfaction with certain clerkships by entering them into a database. They are allotted a certain number of points, and they distribute these points among specific assignments. Additionally, the students specify whether time and/or location are important to them. The system tries to assign specific choices with the highest happiness values, granting time and/or location if either is specified as important. For the rest of the assignments, it picks random students, and tries to make switches between schedules that have a small impact on the satisfaction.

Essentially, the system DMS is implementing now is an automation of the process used in past years. While this may make the process easier for the students and the hospitals, it will not necessarily produce better results. The problem is rich in complexity, and needs to be analyzed in a pure theoretical fashion. By understanding the structure of the complexity, perhaps we can make an algorithm that will not only run efficiently, but also increase student satisfaction. We would also like to be fair to all students; we do not want to rely on random numbers to determine preference.

This paper discusses the complexity of clerkship scheduling; we do not attempt to solve the specific problem of student scheduling at DMS, but rather we abstract clerkship scheduling to a combinatorial scheduling problem. A system that is specific to one medical school could be designed using the ideas from this abstraction. We rigorously define the most general problem, prove it to be NP-hard, and provide an approximation algorithm that produces excellent results. An implementation of this algorithm is also described, and its performance is tested and analyzed.

It is divided into four sections:

•Section A focuses on placing the central problem of clerkship scheduling in a theoretical framework. General clerkship scheduling is strictly defined using set notation. Subproblems are either solved by polynomial-time algorithms or proven NP-hard.

•Section B discusses how to come up with an approximation algorithm for general clerkship scheduling, as defined in section A. Upper bounds on student happiness are defined, proven and analyzed. A general methodology is given for using these upper bounds to produce a schedule, using exponential-time algorithms as examples.

•Section C defines and analyzes the *Flatten* algorithm, a polynomial-time approximation algorithm for general clerkship scheduling that uses the methodology described in section B.

•Section D describes an implementation of the Flatten algorithm, and discusses its performance in terms of the upper bounds described in section B, for instances that would occur in practice. Essentially, this is an implementation of the Flatten algorithm. However, this implementation also considers one issue specific to DMS concerning a *block* structure of clerkship assignments imposed on each student.

Source code may be found at: http://www.dartmouth.edu/~jonfeld.

A. Complexity of Subproblems

How do we quantify student satisfaction in a general way? Some students may want specific clerkships, times and locations. Others may want to be in a certain location at a certain time, or at a certain location for a certain clerkship. Maybe some students have seniority over others. Maybe some students get precedence for certain clerkships due to special ability or training.

In all of these cases, we can express student satisfaction by assigning a discrete value to the overall *happiness* a certain assignment will produce. These values can be represented by an array of numbers, measuring happiness for each particular assignment to a clerkship, time and location. Hospital capacities can be similarly described, by an array of values representing how many students are allowed to be scheduled to a particular clerkship, time and location.

In this section, we will concentrate on solving the problem of using the values in these arrays to come up with a schedule that assigns each student to each clerkship at a unique time, and obeys the hospital capacities. The means by which these values are produced can be left as a policy decision for the medical schools.

We can think of a schedule as a collection of assignments, where each assignment is a particular student, clerkship, time and location. To have a *legal* schedule for all students, each student must be scheduled to each clerkship exactly once. Additionally, students may not have time conflicts, so no more than one clerkship can be scheduled for a specific student and time. Finally, the collection of these assignments may not violate the capacity constraints of the hospital, so no more than the capacity may be scheduled at a specific clerkship, time and location.

Notation And Definitions

To talk about instances of clerkship scheduling abstractly, I will use the following notation and definitions:

Terms

•A **spot** is a specific clerkship, time and location to which students are scheduled.
•An **assignment** is a placement of a particular student into a particular **spot**.
•The **capacity** of a **spot** is the maximum number of assignments allowed to that **spot**.
•A **clerkship/time assignment** is a potential assignment of a particular student to a clerkship and a time, at some arbitrary location.
•A **schedule** is a set of assignments for all students.
•An **individual schedule** is a set of assignments for one student.
•An **individual optimal schedule** for a specific student is the individual schedule with the highest total happiness value that assigns the student to each clerkship at unique times (with or without unique locations), ignoring capacities.
•The **unique location constraint** is the requirement that each assignment for a particular student be to a unique location.

Sets and variables

• N = set of all possible students.
• I = set of all possible times.
• J = set of all possible clerkships.
• L = set of all possible locations.
• S = set of all spots = I x J x L.
• A = set of all possible assignments = N x I x J x L.
• c(s) = capacity of spot s, where $s \in S$.
c_{ijl} = capacity of spot (i, j, l).

• h(a) = happiness with assignment a, where $a \in A$.
h_{nijl} = happiness with assignment (n, i, j, l).

Instances
•An instance of clerkship scheduling will be described by the sizes of the sets N, I, J and L, surrounded by brackets. For example, **{|N| = 5, |I| = 10, |J| = 10, |L| = 1}** represents an instance with 5 students, 10 time slots per clerkship, 10 clerkships, and one location per clerkship/time assignment.
•**The general problem** is {|N| arbitrary, |I| arbitrary, |J| arbitrary, |L| arbitrary}, without the unique location constraint.

Formal Definition of the general problem:
Clerkship Scheduling
Given:
•Set of spots S = I x J x L, where I, J and L are sets containing possible times, clerkships, and locations, each of arbitrary size;
• $\forall\ s \in S$, capacity c(s).
•Set of students N, of arbitrary size;
•Set of possible assignments A = N x I x J x L;
• $\forall a \in A$, happiness h(a).

Question:
Find $\mathbf{H_{max}}$, the maximum happiness over all legal schedules, where
• A legal schedule is a set of assignments $A' \subseteq A$ s.t.
a) $\forall$ student/clerkship pairs $(n,j) \in (N \times J)\ \exists$ exactly one $a \in A'$ s.t. a agrees with that pair in both n and j,
b) $\forall$ student/time pairs $(n,i) \in (N \times I)\ \exists$ no more than one $a \in A'$ s.t. that a agrees with that pair in both n and i, and
c) $\forall\ s \in S,\ \exists$ no more than c(s) $a \in A'$ s.t. s agrees with a in i, j and l.
• The happiness of a schedule $H(A') = \sum_{a \in A'} h(a)$.

A1 Finding the Individual Optimal Schedule

The problem of finding the individual optimal schedule for one student with only one location per clerkship can be simply reduced to a weighted bipartite matching problem. The basic idea is to make a vertex for each clerkship, a vertex for each time, and transform the happiness values to weighted edges between clerkship and time vertices.

Weighted Bipartite Matching:
An undirected bipartite graph is a graph G(V,E) whose vertices can be partitioned into two disjoint subsets V_1 and V_2 s.t. $\forall$ edges $\{u,v\} \in E$, $u \in V_1$ and $v \in V_2$.
Given an undirected bipartite graph G(V,E) and $\forall\ e \in E$, a weight w(e), find the *matching* with the maximum weight, where a matching is a set of edges $E' \subseteq E$ s.t. $\forall$ vertices $v \in V$, at most one $e \in E$ is incident to v, and the weight of the matching = $\sum_{e \in E'} w(e)$.

Algorithm A1: Find an individual optimal schedule using Bipartite Weighted Matching:

Given: •N, I, J, L, sets of students, times, clerkships and locations.
•|N| = 1, |L| = 1.
•A = set of all possible assignments = N x I x J x L.
•S = set of all possible spots = I x J x L.
•h_{nijl} = happiness with assignment (n, i, j, l).

Return:•A'⊆A, where A' is an individual optimal schedule.

•Using the values in h_{1ij1}, we construct an instance of bipartite weighted matching:

$\forall j \in J$, make a vertex x in V_1 (x_1, x_2, ... $x_{|J|}$). $\forall i \in I$, make a vertex y in V_2(y_1, y_2, ... $y_{|I|}$). Create a complete bipartite graph by placing an edge from each vertex in V_1 to each vertex in V_2, setting w(j,i) = h_{1ij1} + k, where w(j,i) = the weight of an undirected edge $\{x_j, y_i\}$ and k is a constant > 0.

The capacities c_{ijl} can be ignored, since we are only scheduling one student to their individual optimal schedule.

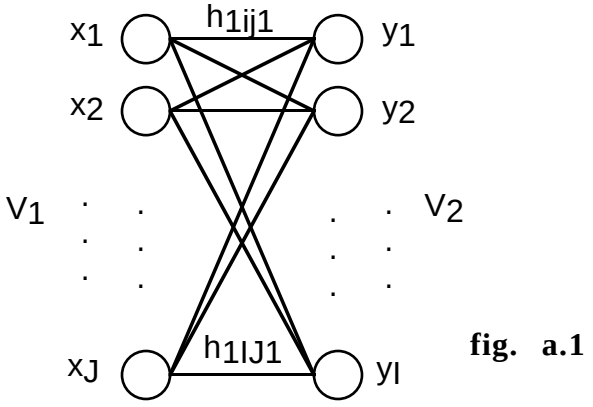


fig. a.1

Proof of the Correctness of Algorithm A1:

Lemma A1.1: |E'| ≤ |J|, where E' is the set of edges found by the maximum weight matching.
Proof: Assume |E'| > |J|, so there are > |J| edges in the matching. Every edge has to connect V_1 to V_2, and edges cannot share vertices, there must be more than |J| vertices in V_1. There are only |J| vertices in V_1, therefore the assumption is false.

Lemma A1.2: |E'| ≥ |J|.
Proof: Assume |E'| < |J|, so there are < |J| edges in the matching. There must exist a vertex $u \in V_1$ that is not on an edge in the matching, since $|V_1| = |J|$. There must also exist a vertex v in V_2 that is not on an edge in the matching, since |I| ≥ |J|, and therefore $|V_2| \geq |V_1|$. If we add edge {u,v} to the matching (it must exist, since the graph is complete), the weight of the matching will increase by at least the constant k, hence the original matching was not of maximum weight, which is false, and therefore the assumption is false.

It follows that |E'| = |J|.

Lemma A1.3: If A' = the set of all assignments (1,i,j,1) s.t. $\{x_j, y_i\} \in E'$, then A' is a legal schedule.

Proof: A' meets the three criteria for being a legal schedule:

- $\forall$ fixed j, $\exists$ exactly one (1,i,j,1) $\in$ A'. This is true because |E'| = |J|, so every x_j is included in exactly one edge $\{x_j, y_i\} \in E'$, and therefore $\forall$ fixed j, assignment (1,i,j,1) is the one and only assignment included in A'.
- $\forall$ fixed i, $\exists \leq$ one j for which assignment (1,i,j,1) $\in$ A'. This is true because $\forall \{x_j, y_i\} \in E'$, only (1,i,j,1) $\in$ A', and all y_i are included in no more than one edge, since edges cannot share vertices.
- Capacities are ignored.

The weight of the matching corresponds to the happiness of the students, so a higher weight matching will yield a schedule with a higher total happiness. Since the maximum weight matching E' will always yield a legal schedule A', the maximum weight matching will yield the legal schedule A' with maximum happiness H_{max}. Therefore $H(A') = H_{max}$.

◊

A2 Location As The Third Dimension

What if we add locations? Happiness values are now specific to all possible clerkship/time/location assignments. How does this complicate the problem of finding the best schedule for one student?

It makes a big difference if students are required to go to a unique location for every assignment. If this is required, finding the optimal solution for one student is much more difficult, and is actually NP-hard with respect to |I|,|J| and |L|. This is clear with a reduction from 3-Dimensional Matching:

Reduction A2: Reduction from 3DM to {|N| = 1, |I| arbitrary, |J| arbitrary, |L| arbitrary}, with the unique location constraint.

Black box for one student with unique locations::
Assume the existence of a black box that when given a set of happiness values h_{nijl}, returns A', a legal schedule (with unique times *and* locations), maximizing H(A'), the total happiness. Again, the capacities are ignored, because |N| = 1.

- *Three Dimensional Matching(3DM)* [1]

Disjoint sets X,Y,Z, all with exactly q elements. M is a set of triples $(x,y,z) \in X \times Y \times Z$. Does M contain a matching; Does $\exists$ $M' \subseteq M$ s.t. |M'| = q and no two elements of M' agree in any coordinate?

Reduce a general instance of 3DM to a specific instance for the black box:

- Set |I| = |J| = |L| = q.
- $\forall$ triples $(x,y,z) \in M$, set $h_{1xyz} = k$, where k is a constant > 0.
- $\forall$ triples $(x,y,z) \notin M$, set $h_{1xyz} = 0$.
- Feed the happiness values into the black box and find the schedule A' with happiness $H(A') = H_{max}$.

Proof of Reduction A2:

Lemma A2.1: $H_{max} = kq$ iff M contains a matching.
Proof:
a) If $H_{max} = kq$, then M contains a matching.
Every legal schedule has exactly |J| assignments (1,i,j,l). $\forall i,j,l\ h_{nijl} = 0$ or $h_{nijl} = k$. |J| = q, so if H(A') = kq, that means $\forall i,j,l$ s.t. $(1,i,j,l) \in A'$, $h_{nijl} = k$. If $h_{nijl} = k$, $(x,y,z) \in M$; therefore $\forall i,j,l$ s.t. $(1,i,j,l) \in A'$, $(i,j,l) \in M$. A legal schedule has unique clerkships, times *and locations* (with the unique location constraint) for all assignments, so $\forall a \in A'$, a is unique in every coordinate.

$\exists\ M' \subseteq M$ s.t. |M'| = q, and no two elements of M' agree in any coordinate, since $\forall i,j,l$ s.t. $(1,i,j,l) \in A'$, $(i,j,l) \in M$, and |A'| = |J| = q. Therefore, M contains a matching.

b) If M contains a matching, then $H_{max} = kq$.

If $\exists$ a matching in M, then $\exists\ M' \subseteq M$ s.t. |M'| = q and no two elements of M' agree in any coordinate. Let set of assignments $A'' \subseteq A$ consist of all assignments (1,x,y,z) s.t. $(x,y,z) \in M'$. $\forall (x,y,z) \in M$, $h_{1xyz} = k$, therefore H(A") = k |A"|.

A" includes a different assignment for every unique $(x,y,z) \in M'$, and no two elements of M' agree in any coordinate. |M'| = q = |J|, therefore, A" includes |J| assignments that are unique in every coordinate. It follows that |A"| = |J|, and that A" is a legal schedule, since A" includes |J| assignments, unique in every time, clerkship, and location. H(A") = k|A"| = k|J| = kq. Therefore $H_{max} \geq kq$.

Consider all legal schedules $B \subseteq A$:

- |B| = |J|, since every clerkship must assigned exactly once.
- $H(B) \leq k|J|$, since $\forall a \in A$, h(a) = 0 or h(a) = k.

H_{max} represents the total happiness value H(B) for some legal schedule $B \subseteq A$, therefore $H_{max} \leq k|J|$. |J| = q, so $H_{max} \leq kq$. $H_{max} \geq kq$, therefore $H_{max} = kq$.
◊

A3 Lifting The Unique Location Restriction

Without the unique location restriction the problem for one student is much easier. It is essentially no different than having only one location, since every student will be happiest in the best location for each clerkship/time assignment. We simply have to isolate the best location for each clerkship/time assignment, and use bipartite weighted matching on those location assignments.

Algorithm A3: Polynomial time algorithm for {|N| = 1, |I| arbitrary, |J| arbitrary, |L| arbitrary}, without the unique location restriction.

Given: •N, I, J, L, sets of students, times, clerkships and locations.
- |N| = 1
- A = set of all possible assignments = N x I x J x L.
- S = set of all possible spots = I x J x L.

- c_{ijl} = capacity of spot (i, j, l).
- h_{nijl} = happiness with assignment (n, i, j, l).

Return:•A'⊆A, where A' is an individual optimal schedule.

•Let $A_1 \subseteq A$ = all assignments (n,i,j,l) s.t. $h_{nijl} = \underset{1 \le l \le L}{Max}(h_{nijl})$; A_1 is the set of |I||J| location assignments with the highest happiness values ∀ fixed i,j.
•Use A_1 as the complete assignment set in an instance of finding an individual optimal schedule for one location, as described in **(A1)**.
•Schedule $A' \subseteq A_1 \subseteq A$ is returned by the instance described in **(A1)**.

Proof of Correctness of Algorithm A3:

Lemma A3.1: A' is the optimal schedule ∀ assignments $a \in A_1$.
Proof: The matching reduction described in **(A1)** finds the optimal schedule for instances with only one location per clerkship/time assignment. This instance described by A_1 has only one location per clerkship/time assignment, therefore A' is optimal ∀ assignments $a \in A_1$.

Lemma A3.2: The optimal schedule A' $\forall a \in A_1$ is also optimal ∀ $a \in A$.
Proof by contradiction:

Assume ∃ optimal schedule $B \subseteq A$ s.t. H(B) > H(A'). Create a new schedule $B' \subseteq A_1$ whose assignments all agree with assignments in B in both i and j. $H(B') \geq H(B)$, since A_1 is the set of |I||J| location assignments with the highest happiness values ∀ fixed i,j. A' is the optimal schedule ∀ $a \in A_1$ **(A3.1)**, therefore $H(A') \geq H(B') \geq H(B)$, and our assumption is false.
◊

A4 Multiple Students As The Next Dimension

So far the problems have been restricted to one student, but the real complexity of clerkship scheduling occurs when there an arbitrary number of students. In fact, the number of students will be the largest variable in practice, since the number of clerkships, times and locations will usually all be fixed. For problems with one student, we could ignore the capacities, but now the capacities constrain the individual student. To find the optimal solution, we have to resolve complex relationships between student preferences and capacities.

The most general case of clerkship scheduling I will consider is {|N| arbitrary, |I| arbitrary, |J| arbitrary, |L| arbitrary}, without the requirement of going to a unique location for every assignment. The happiness values are now specific to four dimensions: n,i,j and l. The capacities are specific to three dimensions: i,j and l . The general problem is NP-hard, even when |J| = 1, and/or all capacities = 1. To prove it NP-hard, we again reduce from 3DM:

Reduction A4: Reduction from 3DM to {|N| arbitrary, |I| arbitrary, |J| arbitrary, |L| arbitrary}, without the unique location restriction:

Black box for one student with unique locations:
Assume the existence of a black box that when given A, a set of assignments, happiness values h_{nijl} and capacities c_{ijl} returns A', a legal schedule, where $H(A') = H_{max}$.

Three Dimensional Matching(3DM)
Disjoint sets X,Y,Z, all with exactly q elements. S is a set of triples $(x,y,z) \in (X \times Y \times Z)$. Does S contain a matching; Does ∃ $S' \subseteq S$ s.t. |S'| = q and no two elements of S' agree in any coordinate?

Reduce a general instance of 3DM to a specific instance for the black box:

- Set N = I = J = L = q
- $\forall$ $(x,y,z) \in S$, set $h_{xy1z} = k$.
- $\forall$ $(x,y,z) \notin S$, set $h_{xy1z} = 0$.
- Set all capacities $c_{ijl} = 1$
- Feed A, the happiness values h_{nijl} and capacities c_{ijl} into the black box and find the best schedule A'.

Proof of Reduction A4:

To prove the reduction, we must show that H(A') = kq iff S contains a matching

a) If H(A') = kq, then S contains a matching.

All $h_{nijl} = 0$, except those for j = 1. So, the weight of an individual student's schedule depends completely on when and where they are assigned for j = 1.

Lemma A4.1 : If H(A') = kq, then there are q occurrences of a j = 1 assignment (an assignment to clerkship 1 for some student n, time i, location l), all of which don't agree in any n, i or j, and all have $h_{nijl} = k$.
Proof: The only possible happiness values for a j = 1 assignment are k and 0. Therefore, if H(A') = kq, then $\exists$ exactly q occurrences of a j = 1 assignment with $h_{nijl} = k$. None of these q assignments will agree in any i or l, since all $c_{ijl} = 1$. Furthermore, these q assignments will all be for different students (and therefore won't agree in any n), since each student has exactly one j = 1 assignment, and there are q students.

If H(A') = kq, then there are q occurrences of a j = 1 assignment, all of which don't agree in any n, i or l, all with happiness k, as shown above. If $h_{ni1l} = k$, then $(n,i,l) \in S$. So, the existence of a schedule with H(A') = kq suggests the existence of q members of S, all of which don't agree in any n,i or l. Therefore, S contains a matching.

b) If S contains a matching, H(A') = kq

A4.2 Lemma: If S contains a matching, each student has unique i and l for which they can be assigned to j = 1 with $h_{ni1l} = k$.
Proof: If S contains a matching, there is a subset of q members of S, all of which don't agree in any x,y or z. $\forall$ $(x,y,z) \in S$, $h_{ni1l} = k$. Therefore, there are q happiness values for j = 1, all with a unique n, i and l, all = k. There are q students, so each one has unique i and l for which they can be assigned to j = 1 with happiness k.

A4.3 Lemma: If S contains a matching, H(A') ≥ kq.
Proof: Put all q of the j = 1 assignments described above into a new schedule $B \subseteq A$. B is legal for j = 1, with H(B) ≥ kq, since $c_{ijl} = 1$ $\forall$ i, l, and **(A4.2)** holds. All other assignments in B are included arbitrarily; that is, each student's schedule is filled in for all j > 1 in any legal combination. Retaining the legality of B is possible, since all students can use every possible clerkship/time assignment for j > 1 (there are q possible locations for every clerkship/time assignment, and only q students).

Schedule $B \subseteq A$ is legal, with H(B) ≥ kq, therefore the maximum happiness schedule H(A') ≥ kq.

A4.4 Lemma: $H(A') \leq kq$.
Proof: Assume $\exists$ a legal schedule B with $H > kq$. Since all h_{nijl} are either k or 0, there must be $> q$ scheduled assignments a_{nijl} with $h_{nijl} = k$. All such weights occur when $j = 1$, so there must be $> q$ assignments to $j = 1$. This cannot be the case, since there are only q students, and all have exactly one assignment to $j = 1$. Therefore the assumption is false, and $H(A') \leq kq$.

$H(A') \geq kq$ and $H(A') \leq kq$, therefore $H(A') = kq$.
◊

B. Using an Upper Bound To Approximate

Thus the general problem is NP complete. This is not good news to medical school administrators, because it means they will probably never be able to satisfy as many students as they theoretically could. However, all is not lost. A good approximation algorithm could yield excellent results. To find an approximation algorithm, we need to know what heuristics we can use to simplify the problem, and how we can measure the success of these heuristics in theory and in practice.

B1 An Upper Bound

How can we measure the success of an approximation? We want to see how close an algorithm gets to the optimal happiness H_{max}. For example, if the algorithm produces $H = \frac{5}{8}(H_{max})$, it is measured by the coefficient $\frac{5}{8}$. In general, we call this coefficient ρ. Unfortunately, we cannot find the optimal schedule, so we don't know the value of H_{max} with which to calculate ρ. We need an upper bound on H_{max} that is as small as possible to get the truest values of ρ.

B1 The upper bound U.
We can find one student's optimal schedule in polynomial time, if we ignore the capacities, and all the other students (A3). So, if we find the optimal schedule for each student individually, and schedule them in the general problem, this schedule will be made up entirely of optimal schedules. Unfortunately, this schedule is not always legal in the general problem due to capacity constraints. However, the total happiness of this schedule is a nice upper bound for the optimal schedule by which we can judge the success of our approximation algorithms. We will call this upper bound U.

•Let A = N x I x J x L, the set of all possible assignments in an instance of the general problem.
•Let O be the set of all individual optimal schedules found by ignoring the capacities (A3).
•Let A' be the solution to the general problem, so $H(A') = H_{max}$.
•Let $U = \sum_{O_n \in O} H(O_n)$.

The following lemma will show that U is an upper bound for H_{max}:

B1.1 Lemma: $\forall$ instances of the general problem, $H_{max} \leq U$.
Proof: Assume $H_{max} > U$. Therefore, $H_{max} > \sum_{O_n \in O} H(O_n)$. Divide A' into |N| distinct subsets O'k, where $O'_k = \{a_{nijl} \in A' : n = k\}$. These subsets represent the individual schedules for the

solution A'. $\sum_{O'_n \in A} H(O'_n)$ = H(A') = H_{max}, since {O'1 ∪ O'2, ∪…∪ O'|N|} = A'.

Therefore: $\sum_{O_n \in O} H(O_n) > \sum_{O'_n \in A} H(O'_n)$. It follows that ∃ some q for which H(O'q) > H(Oq), where Oq ∈ O. However, O is the set of all individual optimal schedules, therefore H(O'q) ≤ H(Oq). We have a contradiction, therefore our assumption is false.

◊

How close is U to the true H_{max}? In some cases, U = H_{max}:

B1.2 Lemma: ∃ an instance of the general problem for which U = H_{max}.
Proof: Consider a case where all c_{ijl} = |N|. Schedule every student to O, their individual optimal schedule. This schedule is legal, since every spot can hold every student and still be at or under capacity. H_{max} = $\sum_{O_n \in O} H(O_n)$ = U.

◊

In **(B1.2)**, we see a case where $\frac{U}{H_{max}} = 1$. However, this ratio could be as bad as |N|:

B1.3 Lemma: ∃ an instance of the general problem for which $\frac{U}{H_{max}}$ = |N|.

Proof: Consider the following case:

- All c_{ijl} = |N|, except c_{111} = 1.
- All h_{nijl} = 0, except ∀n, h_{n111} = k.

U = $\sum_{O_n \in O} H(O_n)$ = k|N|, since all students' optimal schedules will include their only non-zero happiness value at a_{n111}. However, H_{max} = k, since c_{111} = 1, and only one student can actually be scheduled to their only non-zero happiness value.

$$\frac{U}{H_{max}} = \frac{k|N|}{k} = |N|.$$

◊

So U is not consistently close to H_{max}; U can be equal to H_{max}, and U can be significantly larger than H_{max}. However, in practice, U is still useful. It takes an extreme example to make U significantly larger than H_{max}, one that has a lot of high happiness values in spots with small capacities.

B2 Methodically Rescheduling From The Upper Bound

What heuristics can we use? Our final schedule has to get every student into every clerkship, and it must obey the capacity constraints, so our approximation must yield a legal schedule. In addition, we want to get as close to the upper bounds as we can. Instead of searching within the legal schedules, we will start with one of the upper bounds, and work our way down until we find a legal schedule.

What if we start with the upper bound U? To get a schedule with total happiness of U is simple; all we have to do is schedule all students to their individual optimal schedule, which is a

polynomial time problem, as shown in **(A3)**. This schedule is not necessarily legal, so to work our way down to a legal schedule, we have to methodically reschedule the students, trying to keep the happiness up while still working towards a legal schedule.

We will first look at a solution to the general problem that finds the optimal schedule in exponential time, but illustrates the concept of using the upper bound as a starting point:

Algorithm B2.1: Exponential time algorithm to solve the general problem:

Given: •N, I, J, L, sets of students, times, clerkships and locations.
•A = set of all possible assignments = N x I x J x L.
•S = set of all possible spots = I x J x L.
•c_{ijl} = capacity of spot (i, j, l).
•h_{nijl} = happiness with assignment (n, i, j, l).

Return:•An optimal legal schedule.

• $\forall$n, set A^o_n = the set of all subsets A'$\subseteq$A s.t. A' schedules student n to exactly |J| different clerkships, all at unique times; A^o_n is the set of all legal individual schedules for student n.
•Let set P = (A^o_1 x A^o_2 x A^o_3 x ... x $A^o_{|N|-1}$ x $A^o_{|N|}$) ; P is the set of all possible combinations of legal individual schedules.
• $\forall p \in P$, let q(p) = ($A_1 \cup A_2 \cup \ldots \cup A_{|N|}$), where p = ($A_1$, A_2, ... $A_{|N|}$). Each q(p) represents the total schedule including all the individual schedules in p.
•Let set Q = {q(p) : $p \in P$}.
•Return q' = $\arg\max_{legal \; q \in Q} H(q)$, the legal schedule in Q with the highest happiness value.

Running Time of Algorithm B2.1:

This algorithm is obviously exponential, since it must look at every possible combination of every possible schedule for all |N| students, and test for legality.

• $\forall$n, Contstruct A^o_n:	
Find all legal individual schedules for all students	O(|N||L||I|!)
•Construct P,Q:	$O(|L||I|!)^{|N|}$
• $\forall q \in Q$, test for legality	$O(|L||I|!)^{|N|}$
Total time needed:	**$O(|L||I|!)^{|N|}$**

Algorithm **B2.1** takes an exponential amount of time because it looks at every possible combination of schedules. We can improve upon this method by constructing solutions greedily; we sort the individual schedules before we combine them. When we combine them, we will test them for legality. We will construct complete schedules in descending order of happiness, so we can stop after finding our first legal schedule. Thus we will find an optimal schedule with a greedy algorithm:

Algorithm B2.2: Pseudocode for a greedy algorithm to find a solution to the general problem

Given and Return values: same as **B2.1**

For all students $n \leq |N|$
    let A'_{nk} = the schedule in A^{o}_{n} with the kth highest happiness value.
    let $r_n = 1$, where r_n = the current schedule for student n being considered.

Begin Loop:
    Let B' = empty schedule, therefore H(B') = 0
    Let changed_student = 0

    /* The following loop tries every schedule where exactly one r_n increases by one, and the schedule with the highest happiness value is kept. */
    For student k = 1 to |N|
        Consider schedule $B = (A'_{1r_1} \cup A'_{2r_2} \cup \ldots \; \mathbf{A'_{k(r_k+1)}} \; \ldots \cup A'_{|N|r_{|N|}})$
        If H(B) > H(B'), then let B' = B
        changed_student = k
    Increase $r_{changed\_student}$ by 1

Loop until B' is a legal schedule
Output B'

Proof that Algorithm B2.2 gives an optimal solution:

By contradiction: assume B' is not optimal. Therefore $\exists$ a legal schedule D' s.t. H(D') > H(B'). The algorithm found B' as the first legal schedule. Therefore, all schedules that include individual schedules for any student n with higher happiness values than the individual schedule for student n included in B' must not be legal, or else the algorithm would have chosen it earlier in the loop. So, D' must consist entirely of individual schedules with happiness values greater than or equal to the individual schedules included in B'. Therefore $H(D') \leq H(B')$, and our assumption is false.

◊

In the worst case, Algorithm **B2.2** will still be exponential. This is clear, since there might be only a few possible combinations of legal schedules, all of which have low happiness values; the algorithm would then have to consider most of the combinations of schedules before finding a legal one. Again, more bad news for the medical school administrator.

Algorithm **B2.2** is based on starting with a schedule that is not necessarily legal, but has a happiness value equal to the upper bound U. It proceeds by using a heuristic to change that schedule until it is legal. We can use this general concept, but employ a polynomial time heuristic, so we will be able to find good solutions quickly, with happiness values close to the upper bounds.

C. The Flatten Algorithm

We now introduce an approximation algorithm based on the concept of methodically rescheduling from the upper bound **(B2)**. First, some background is given concerning the theory behind the algorithm **(C1)**. Then, pseudocode for the Flatten algorithm is given, along with a proof of correctness and an analysis of the running time **(C2)**. We then provide some theory behind the *tightness* of an instance of scheduling, a phenomenon that seems to affect the running time of the algorithm, as well as the happiness of the schedule it produces **(C3)**. Finally, we explore the limitations of the Flatten algorithm, and provide some possible solutions for getting around these limitations **(C4)**.

C1 Introduction

The Flatten algorithm begins with the upper bound U, and schedules students to each of their individual optimal schedules. If this schedule is legal, the algorithm is done. If it is not legal, it "flattens" the schedule by rescheduling students from spots that are over-capacity, until there exist no such spots.

This seems easy in theory, but it is far from trivial to come up with a good way to do this methodically, so that the algorithm finds a legal schedule in a reasonable amount of time. We need a heuristic to reschedule students away from individual over-capacity spots, and into other spots. This heuristic must reduce the over-capacity spots to capacity, maintain as high a happiness as possible, and eliminate more and more options to switch as it progresses, so it may approach a legal schedule.

I have found such a heuristic based on the concept in **B2.2** of using happiness rankings of schedules specific to each student. The Flatten algorithm uses happiness rankings of *location assignments,* specific to each student/clerkship/time.

C2 The Heuristic

Flattening the schedule is the heuristc that enables us to move students away from over-capacity spots. The Algorithm first searches for the most over-capacity spot. When an over-capacity spot is selected, exactly as many students are rescheduled from that spot as are needed to make that spot obey its capacity. The students selected to be rescheduled are the ones for whom moving them would cause the least reduction in happiness value. When students are selected to be rescheduled, they are never allowed to return to that spot.

How do we reschedule the selected students, and force them never to return to spots from which they are rescheduled? Consider the algorithm described in **(A3)** that found an individual optimal schedule for one student with multiple locations. The problem reduced to an instance of finding an individual optimal schedule for one location **(A1)** by including only the location assignments to clerkship j, time i, with the highest happiness values $\forall$ fixed i,j. In fact, we can create an instance of **(A1)** using any location assignments we choose, and **(A1)** will return the optimal schedule among *those* location assignments.

In the Flatten Algorithm, for each clerkship/time assignment, we will maintain the sorted happiness values for each location, and use an array of global variables keep track of which location rank the algorithm is currently considering for each student/clerkship/time; at the beginning of the algorithm, these global variables are all set to 1, since the algorithm begins by scheduling each student to an individual optimal schedule. Each time a student is selected to be rescheduled, we increase the location rank variable for that spot by 1, and create an instance of **(A1)** using the location assignments referred to by the values in the location rank variables.

Thus the selected students are rescheduled from the over-capacity spot, and are forced never to return to that spot again. The algorithm proceeds by finding a new over-capacity spot, and repeating the whole process. As students are restricted further and further by having their location rank variables increased, the schedule is forced towards legality.

For convenience, since we will not be using formal set notation to explain the flatten algorithm, we will use the constants N,I,J,L to denote the size of sets N,I,J,L.

C2.1 Pseudocode for the Flatten Algorithm

Given: N = number of students, I = number of time slots,
J = number of clerkships, L = number of locations.
$\mathbf{h_{nijl}}$ = happiness with assignment (n,i,j,l)
$\mathbf{c_{ijl}}$ = capacity of slot (i,j,l)

Return: A legal schedule A'

Function: **Sort_Student_Happiness**
For all students n
For all clerkship/time assignments i,j
1: •sort h_{nijl} in descending order, and store the original locations of each assignment into a separate array $\mathbf{loc_{nijr}}$. h_{nijr} now represents the rth highest happiness value for student n, time i and clerkship j.
2: •set all $\mathbf{rank_{nij}} = 1$, where $rank_{nij}$ = the current rank being considered for student n, time i and clerkship j.

Function: **Match(student n)**
3: •Create an instance of finding an individual optimal schedule for one location **(A1):**
Values needed for instances of **(A1)**: Sets $\mathbf{N, I, J, L, h_{nijl}}$.
•Let $\mathbf{N}$ = {n}
•Let $\mathbf{I}$ = {1,2,3....I}
•Let $\mathbf{J}$ = {1,2,3....J)
•Let $\mathbf{L}$ = the set of all $\mathbf{loc_{nij}}(rank_{nij})$ $\forall$i,j ; the locations of the current ranks begin considered for student n, time i, clerkship j.
•Let $\mathbf{h_{nijl}} = h_{nij}(rank_{nij})$

4: •Return the schedule **M** returned by **(A1)**.

Flatten_Algorithm
5: •Call Sort_Student_Happiness
6: • $\forall$n, Call Match(n), and add each returned schedule M to the complete schedule **A'**, initializing $\mathbf{at_{ijl}}$ = number of students scheduled to time i, clerkship j, location l.

7: Loop Until A' obeys all the capacity constraints
8: •Find a spot (i,j,l) where $at_{ijl} - c_{ijl} > 0$. This spot is over-capacity.

9: For each student n scheduled at spot (i,j,l):
10: •Store current individual schedule into **O**, an array of original schedules.
11: •Increase $rank_{nij}$ by 1.
12: •Call Match on student n.
13: •Store returned individual schedule M into **D**, an array of schedules.

14: •Sort D and corresponding O by **H(o) - H(d)**, the difference between the happiness value of the new schedule and the happiness value of the old schedule, where o $\in$ O and d $\in$ D.

15: For the first c_{ijl} schedules in D: (with the c_{ijl} lowest differences in happiness)
16: •Remove original schedule O from A', decreasing at_{ijl} by 1 for all assignments o $\in$ O
17: •Add new schedule D to A', increasing at_{ijl} by 1 for all assignments d $\in$ D
18: For the remaining schedules in D:
19: •Decrease $rank_{nij}$ by 1.
20: Return A'

◊

The Flatten algorithm will not always find a legal schedule if one exists. At line **11**, We increase the value of $rank_{nij}$ by 1. If $rank_{nij} = L$, we would not be able to increase this value and maintain a meaningful value. We say that the student has reached an *empty rank* for this clerkship/time assignment. We will discuss in great detail the situations under which empty ranks occur, as well as solutions to get around them, in section **(C4)**. However, if no students reach any empty ranks, and the flatten algorithm as listed above finds a schedule, we can prove that this schedule is legal:

Lemma C2.2: If no students reach empty ranks, then A', returned by the Flatten algorithm, is a legal schedule.

Proof:

•Every individual schedule added to A' during the course of the algorithm, at lines **6** and **17**, has been returned by the Match function. The Match function returns only individual optimal schedules, which are legal.

•All students are initially scheduled to A' exactly once at line **6**, and every time a student is removed from A' at line **16**, they are immediately placed back into A' at line **17**. Therefore, when the algorithm terminates, all students are scheduled in A' exactly once.

•Since all students are scheduled in A' exactly once, and every individual schedule added during the course of the algorithm is legal, A' contains exactly one legal individual schedule for each student.

•Line **7** is the start of a loop that does not terminate until A' obeys all the capacity constraints. Since A' contains exactly one legal schedule for each student, and it obeys all the capacity constraints, A' is a legal schedule.

◊

C2.3 Running Time of the Flatten Algorithm

•Function: Sort_Student_Happiness
  Loop NIJ times, sorting L items
    time: NIJLlgL
  Total time: NIJLlgL
    I ≥ J, so time = **$O(NI^2LlgL)$**

•Function: Match(student n)
  Transform IJ happiness values into an instance of bipartite weighted matching
    time: IJ
  Run a bipartite weighted matching, #Vertices = I + J, #Edges = IJ
    time: (I + J)IJ
  Total time: IJ + IJ(I + J).
    I ≥ J, so time = **$O(I^3)$**

•Flatten Algorithm

•Call Sort_Student_Happiness	**$O(NI^2LlgL)$**
•Call Match on every student	**$O(NI^3)$**
•Add returned schedules to A' for every student (each schedule size IJL, I ≥ J)	**$O(NI^2L)$**
•Loop until A' is legal; let **P** = #of iterations	
•Find an over-capacity spot	**$O(PI^2L)$**

(IJL spots,I ≥ J)
•For each student scheduled at that spot (≤**N**)
•Store current schedule **O(PNI^{2}L)**
(each schedule size IJL, I ≥ J)
•Call Match on each student **O(PNI^3)**
•Store returned schedule **O(PNI^{2}L)**
(each schedule size IJL, I ≥ J)
•Sort schedules **O(PNlgN)**
(≤N schedules)
•Remove and add schedules **O(PNI^{2}L)**
(≤ N schedules, each schedule size IJL, I ≥ J)

TOTAL = O(PNI^3 + PNI^{2}L + NI^{2}LlgL + PNlgN)
= O(P(NI^3 + NI^{2}L + NlgN) + NI^{2}LlgL)

Since N is large compared to I,J and L, it is important to look at the degree of N independent of I,J and L in the analysis of the running time. PNlgN represents the most significant term of the total running time, so we can say that the algorithm runs in **O(PNlgN)**. But, even though I,J and L will be small compared to N, they will still be significantly sized numbers. I will discuss running time in practice when I describe testing of the implementation in section D.

How big is P? The size of P will vary according to how many times we need to flatten a spot before we find a legal schedule. P = the number of iterations of the flattening loop, which begins at line **7**. At each iteration of the flattening loop, we permanently increase the value of at least one $\mathbf{rank_{nij}}$. Each $rank_{nij}$ can only take on L different values, corresponding to the L possible locations for each clerkship/time assignment. So, we can only permanently increase the value of at least one $\mathbf{rank_{nij}}$ NIJL times. Therefore P ≤ NIJL. Since I,J and L are small, **P = O(N)**.

Therefore, the theoretical running time of the Flatten algorithm = **O(N^{2}lgN).**

◊

The actual running time of the algorithm is highly dependent upon P, which is not an input value, but rather a result of how many times we need to flatten spots until we reach a legal schedule. Additionally, every time we flatten a spot, we reduce the happiness of the schedule. If we never have to flatten a spot, P could equal 0. We have shown above that P ≤ NIJL.

How can we determine the number of times we need to flatten spots from the input values? Through experimentation with the implementation in section D, I will show a relationship between the *tightness* of the instance, as defined in **(C3)**, and the happiness and running time of the schedule returned by the flatten algorithm.

C3 The Tightness of an Instance

We introduce the concept of the *tightness* of an instance; how many available slots there are compared to how many are needed. Formally,

Definition C3.1: The Tightness of an Instance:

•The *tightness*, T, of instance {|N|,|I|,|J|,|L|}, (with capacities c_{ijl}) = $\frac{\sum_{1\le i\le |I|}\sum_{1\le j\le |J|}\sum_{1\le l\le |L|} C_{ijl}}{|N||J|}$.

If T < 1, there are fewer available assignments than needed assignments; therefore there are no possible legal schedules. When T > 1, there could be a legal schedule, but not necessarily;

Lemma C3.2: ∃ Instances of clerkship scheduling where T > 1, but ∃ no legal schedules.
Proof: Every student must be scheduled to every clerkship. Therefore, if ∃ a legal schedule, then $\forall$j, $\sum_{1\le i\le |I|}\sum_{1\le l\le |L|} C_{ijl} \ge |N|$. Consider an instance of clerkship scheduling where:

- $\sum_{1\le i\le |I|}\sum_{1\le l\le |L|} C_{i1l} = |N| - 1$; the sum of the capacities of the spots for clerkship 1 is |N| - 1.
- $\sum_{1\le i\le |I|}\ \sum_{2\le j\le |J|}\ \sum_{1\le l\le |L|} C_{ijl} = |N||J|$, the sum of the capacities of all other spots is |N||J|.

In this instance, T = $\frac{\sum_{1\le i\le |I|}\sum_{1\le j\le |J|}\sum_{1\le l\le |L|} C_{ijl}}{|N||J|} = \frac{|N||J|+|N|-1}{|N||J|} = 1+\frac{|N|-1}{|N||J|}$. Therefore, T > 1.
However, for j = 1, $\sum_{1\le i\le |I|}\sum_{1\le l\le |L|} C_{ijl} = |N| - 1$. Therefore, ∃ no legal schedules.
◊

Lemma **C3.2** points out that even though the total number of available spots is larger than the spots needed, there must still be a sufficient number of available spots for each clerkship. Now consider instances where |I| = |J|. Since every student must be scheduled to each clerkship at a unique time, and |J|=|I|, every student must also be scheduled to each time. As in **A5.2** for *clerkships*, If such an instance had fewer than |N| available spots for any *time*, it would also have no legal schedules.

Is having enough available spots for every clerkship, and enough available spots for every time a sufficient condition to claim that ∃ a legal schedule? No, in fact it is not:

Lemma C3.3: ∃ an instance of clerkship scheduling for which
- T > 1,
- $\forall$j, $\sum_{1\le i\le |I|}\sum_{1\le l\le |L|} C_{ijl} \ge |N|$, and
- $\forall$i, $\sum_{1\le j\le |J|}\sum_{1\le l\le |L|} C_{ijl} \ge |N|$, but ∃ no legal schedules.

Proof:
Consider the following instance:
- |I| = |J| = |N| = 3
- |L| = 1
- The capacities are represented by the following chart,

$j=$ \ $i=$	1	2	3
1	1	1	1
2	0	3	0
3	2	0	2

fig c3.4

where the rows represent clerkships, the columns represent time slots, and an entry in the table represents c_{ij1}, the capacity of clerkship j, time slot i. We see that $\forall$j, $\sum_{1\le i\le |I|}\sum_{1\le l\le |L|} C_{ijl} \ge |N|$, and $\forall$i, $\sum_{1\le j\le |J|}\sum_{1\le l\le |L|} C_{ijl} \ge |N|$. T = $\frac{\sum_{1\le i\le |I|}\sum_{1\le j\le |J|}\sum_{1\le l\le |L|} C_{ijl}}{|N||J|} = \frac{10}{9}$. Therefore T > 1.

•We claim that $\exists$ no legal schedule for this instance. Proof by counterexample: Assume $\exists$ a legal schedule. All 3 students must be assigned to clerkship 1, at some time slot. Since $C_{i11} = 1$ for all 3 time slots i, each student must be assigned to a different time slot for clerkship 1. Therefore, some **student n must be assigned to clerkship 1, time slot 2**. Student n must also be assigned to clerkship 2, at some time slot. $c_{121} = 0$ and $c_{321} = 0$, so **student n must be assigned to clerkship 2, time slot 2**. The two assignments in boldface violate the requirement that all student assignments be to unique time slots. Therefore, our assumption is false.
◊

C4 Limitations

There is no guarantee that the Flatten algorithm will come up with a legal schedule if one exists. Consider the following situation, which I will refer to as a student reaching an *empty rank:*

C4.1 One example of an *Empty Rank:*
•Some student n has been rescheduled from over-capacity spots a number of times, in particular from locations within clerkship 3, time 5.
•After a while, the algorithm decides to flatten clerkship 3, time 5, location 8. It begins to reschedule students when it comes across student n.
•The algorithm sees that $rank_{nij} = L$, where $rank_{nij}$ = student n's rank for clerkship j,time i. . This means that during the previous iterations of the algorithm, student n has been rescheduled to each location assignment within that clerkship/time
•Student n cannot be rescheduled anywhere for that clerkship/time; they have reached an *empty rank.* at clerkship j, time i.

To get around an empty rank, we can instruct the algorithm to do the following:

C4.2 Method to get around an empty rank:
If spot (i,j,l) is being flattened, and for some student n, $rank_{nij} = L$, and student n is instructed to increase $rank_{nij}$ by 1,

•Let K = a large constant; K > IJL$\left(\underset{i,j,l}{Max}(h_{nijl}) \right)$.

•Call Match on student n, but set the weight of the edge $\{x_j, y_i\} = -K$.

The matching will choose an alternate schedule that doesn't include an assignment to clerkship j, time i. What if an empty rank occurs a number of times for some student n? The algorithm may eventually reach a situation where student n cannot be legally rescheduled at all from a particular spot (i,j,l) using the student's values in $rank_{nij}$. I will refer to this situation as a *dead end*:

C4.3: An Example of a *Dead End:*

•Some student n reaches an empty rank for clerkship 3, time 5.
•Student n is rescheduled a number of times, and reaches a number of empty ranks, especially for clerkship 3.
•The algorithm decides to flatten clerkship 3, time 4, location 2.
•The algorithm tries to reschedule student n, and sees that student n has an empty rank for clerkship 3, time 4. So, it sets the weight to -K, and attempts a matching.
•But, student n has reached an empty rank at all clerkship 3 assignments, and the maximum weight matching just leaves out clerkship 3 from the matching to avoid the -K weight, and therefore returns an illegal schedule; this student has reached a *dead end*.

What should the algorithm do with a student in the case of a dead end? A dead end can be okay if there are enough students at that spot that *can* be rescheduled to flatten the spot. But, if there are more students that have reached this "dead end" than the capacity for the spot, the algorithm cannot proceed, since it will be unable to flatten the spot.

Can we do anything about this situation when it occurs? We can't avoid it without fundamentally changing the algorithm, but we can "clean up" afterwards, by putting the dumped students back into the open spots in the schedule.

How could there be any open spots in the schedule? Every time a student reaches an empty rank for a clerkship/time assignment, it means every spot for that clerkship/time has been flattened. After a spot is flattened, it is left exactly at capacity. If a student is dumped, they have reached enough empty ranks to leave them unable to be scheduled. How could this student possibly be rescheduled, if they have reached so many empty ranks?

After a spot is flattened, it won't necessarily stay at capacity:

Lemma C4.4: Flattened spots do not necessarily stay at capacity:

•Consider some student n who is scheduled to a spot s_1 that is flattened.
•Student n remains at s_1 after it is flattened, since student n is among the students who produce the lowest $c(s_1)$ happiness values when rescheduled, where $c(s_1)$ = the capacity of spot s_1.
•Spot s_1 is now exactly at capacity, since it has just been flattened.
•Later in the algorithm, a different spot s_2 is flattened; student n is also scheduled at s_2.
•Student n is rescheduled from s_2, since rescheduling student n produces a higher happiness value than rescheduling $c(s_2)$ other students, where $c(s_2)$ = the capacity of spot s_2.
•The new schedule for student n does not include s_1, therefore s_1 is now under-capacity.

We take advantage of this phenomenon by removing students that have reached a dead end, and rescheduling these students after the all the spots are all flattened:

C4.5: A method to *Dump* students when they reach a dead end:

During the algorithm, if some spot (i,j,l) is being flattened, and x students scheduled at spot (i,j,l) have reached a dead end, where $x > c_{ijl}$,

- Arbitrarily pick $(x-c_{ijl})$ of the students that have reached a dead end, and *dump* them; take them out of the permanent schedule A', and do not reschedule them.
- Flatten spot (i,j,l) normally, rescheduling the students that have not reached a dead end.
- Proceed with the algorithm, and do not require dumped students to be scheduled when determining the legality of A'.

Dumped students will be ignored for the rest of the algorithm, since students are rescheduled only when they are scheduled in spots to be flattened. Dumped students are not scheduled anywhere, so they will never be put back into A'.

We can build the methods described **C4.2** and **C4.5** into the flatten algorithm, thus allowing the algorithm to handle more cases.

To build **C4.2** into the algorithm, we replace line **11** with the following conditional statement:

9: For each student n scheduled at spot (i,j,l):
10: •Store current individual schedule into **O**, an array of original schedules.
11: **If $\mathbf{rank_{nij} \leq L}$**
•Increase $rank_{nij}$ by 1.
Else
•Let $h_{nij(rank_{nij})} = \mathbf{-K}$, where $K > IJL\left(\underset{i,j,l}{Max}(h_{nijl}) \right)$.
12: •Call Match on student n.
13: •Store returned individual schedule M into **D**, an array of schedules.

For **C4.5**, we replace line **17**, also with a simple conditional statement:

15: For the first c_{ijl} schedules in D: (with the c_{ijl} lowest differences in happiness)
16: •Remove original schedule O from A', decreasing at_{ijl} by 1 for all assignments $o \in O$
17: **If H(D) $\geq$ 0**
•Add new schedule D to A', increasing at_{ijl} by 1 for all assignments $d \in D$
Else
•Let boolean array **Dumped[n]** = TRUE.
18: For the remaining schedules in D:
19: •Decrease $rank_{nij}$ by 1.

Now we must reschedule the dumped students into open spots created by the phenomenon described in **C4.4.** The problem of rescheduling the dumped students is equivalent to an instance of the general problem, since there are an arbitrary number of students, student/spot-specific happiness values, and spot-specific capacities.

We could recurse on the flatten algorithm. However, we must be sure that dumped students are not scheduled to the spots that are already at capacity from the first pass. So, $\forall$ dumped students n, and spots (i,j,l) that are at capacity, we let $h_{nijl} = -K$, as in **(C4.2)**. When the number of dumped students is small enough, an algorithm that finds the optimal legal solution, such as **B2.2** or **B2.3**, may be the best choice.

D. An Implementation of the Flatten Algorithm

I have implemented the flatten algorithm in a way that would be useful in practice to schedule medical students to clerkships at Dartmouth Medical School (DMS). The program is called *Clerk*, and it is written in C++. Clerk is a direct implementation of the Flatten algorithm, with one modification based on a restriction on student schedules imposed by Dartmouth Medical School **(D1)**. *Clerk* also implements the strategy explained in **(C4.5)** to *dump* students when they reach a dead end **(D1.3)**.

D1 Modifications of the Flatten Algorithm

DMS has a requirement that student schedules be in a *block structure*, defined as the following:

D1.1 The DMS *Block Structure*[1]

•There are 3 blocks; each containing 3 clerkships, for a total of 9 clerkships.
•In each block, one clerkship has time length 2, and the other two clerkships have time length 1.
•The clerkships in each block must be scheduled together.
•Within each block, the clerkship with time length two must be scheduled first or last.
•The 3 blocks can be scheduled in any order.

The Block structure has a profound effect on the legality of individual schedules. However, the only part of the flatten algorithm that needs to be changed to accommodate the block structure is the Match function. The Match function outputs the optimal legal individual schedule; this implementation's Match function does the same thing, but obeys the block structure. The flattening process itself remains the same.

In **(C2)** we saw how to control the location ranks for each clerkship/time assignment, and use instances of **(A1)** on certain location assignments. We do the same thing with the Match procedure for the block structure, since the unique location restriction still does not apply. The difference comes in performing the actual bipartite weighted matching, because the block structure forces us to choose only certain sets of assignments.

In this implementation we use a brute force method, and bypass **(A1)** altogether, since the constants are quite small. So, given one location assignment for each clerkship/time, we try every possible individual schedule among those that are legal. How many possibilities are there?

D1.2 The number of individual schedules that obey the block structure

•Let A,B,C represent the three clerkships in one of the three blocks, where A is of length 2, and B and C are both of length 1.
•There are 4 possible clerkship configurations for this block, since A must come first or last:
ABC, ACB, BCA, CBA
•There are 3 blocks, so given the order of the blocks, there are 4^3 clerkship configurations.
•Let X,Y,Z represent the three blocks. There are 6 possible orders of the three blocks:
XYZ, XZY, YXZ, YZX, ZXY, ZYX
•So, the number of possible legal schedules = $6(4^3) = 384$.

Looping through each possibility in the block structure would require 384 happiness calculations. If we tried a similar brute force method for 9 clerkships and 9 time slots without the block structure, there we would need 9! = 362,880 happiness calculations. Even if we use bipartite weighted matching, as in **(A1)**, on 9 clerkships and 9 time slots, the time needed is on the order of (#vertices)(#edges) = 9(81) = 729.

We see from this analysis that to find an individual optimal schedule with the block structure imposed, it is *more* work to use an instance of bipartite weighted matching than to use a brute force method. So the Match function implemented in *Clerk* simply tries every possibility, and returns the legal individual schedule with the highest total happiness.

D1.3 Dumping Students

Clerk also implements the strategy explained in **C4.5** to *dump* students when they reach a dead end. After the Flatten algorithm is complete, *Clerk* reschedules the dumped students:

•Loop through each dumped student n in an arbitrary order:
- •$\forall$ i,j find the location assignment l for which h_{nijl} is maximum, and $c_{ijl} - at_{ijl} > 0$.
- •Call Match on those location assignments, as in (C2)
- •Update values of at_{ijl}.

D2 Performance

We can measure the performance of *Clerk* using the happiness of the output schedule, and the upper bound U, from section **(B1)**. *Clerk* outputs a schedule, A', and it also calculates the total happiness of that schedule, H(A'). U is easy to calculate, because the algorithm begins by scheduling the students so that their happiness equals U. So, when *Clerk* stops running, it outputs a report containing H(A'), U, and ρ, where $\rho = \frac{H(A')}{U}$; ρ represents the happiness of the schedule as a fraction of the upper bound.

There is a close relationship between ρ and T, where T is the tightness of the instance, which we explore through experimentation with *Clerk.* Recall that the tightness of an instance, T, is defined as: $\frac{\sum_{1\le i\le I}\sum_{1\le j\le J}\sum_{1\le l\le L} C_{ijl}}{NJ}$ **(A5).** T compares the number of available spots to the number of spots needed. When T<1, there exist no legal schedules, since there are fewer spots available than spots needed. When T > 1, there could be a legal schedule, but not necessarily, as proven in **(A5.2)** and **(A5.3)**.

We can design our test data to be similar to data that would appear in practice. Students would be given a fixed amount of *points* to distribute arbitrarily among different assignments, indicating their preference for combinations of clerkship, time and location. We simulate this with the following procedure:

- For each student n,
  - Loop until the point total has reached the fixed allocation:
    - Choose a random assignment (n,i,j,l).
    - Place a random happiness value h_{nijl} on that assignment, and increase the point total by h_{nijl}.

As an experimental tool to get a cleaner value for T, we can fix the capacities of the spots: If $\forall$i,j,l, c_{ijl} = CAP, where CAP is a fixed value, $T = \frac{IJL(CAP)}{NJ} = \frac{IL(CAP)}{N}$.

Experiment D2.1: ρ as a function of T, with constant I,J,L and fixed capacities.
•Hold I,J and L constant.
• ∀i,j,l, c_{ijl} = fixed value CAP, as defined above.
•Run *Clerk*, changing the values of N and CAP for each trial.

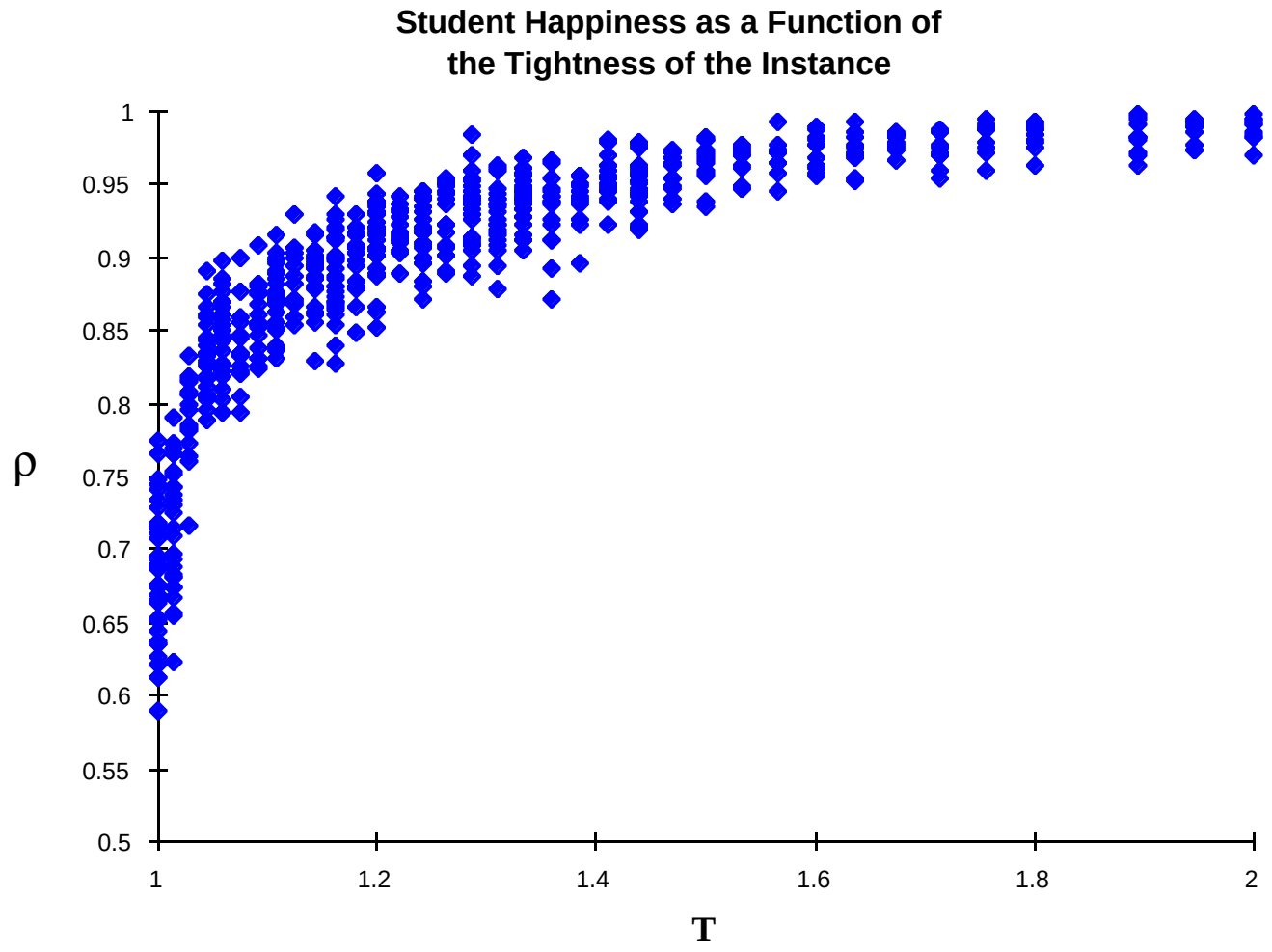
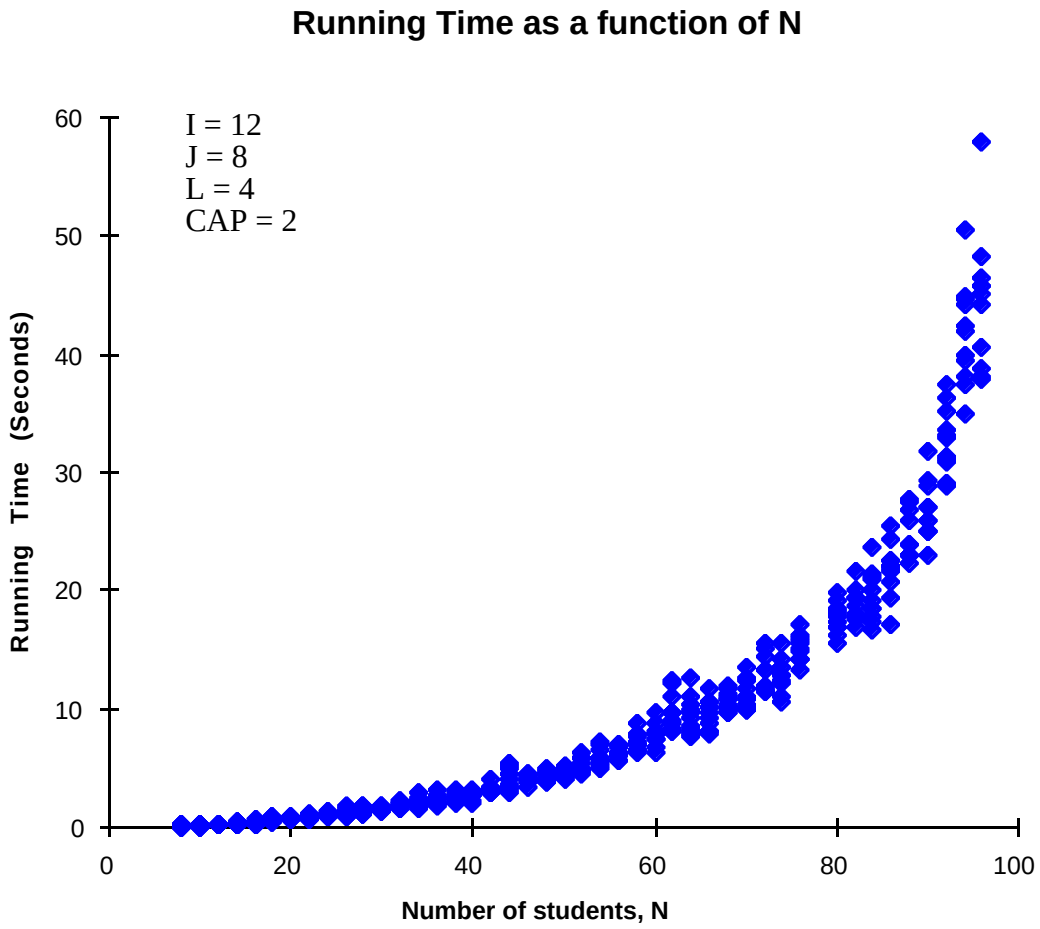


fig d2.1

We see from these data that ρ is a function of the tightness of the schedule. We can also get an idea of the values of ρ *Clerk* can produce. Even for schedules where T = 1, *Clerk* usually produces values for ρ around $\frac{2}{3}$, and never less than $\frac{1}{2}$. When T = 1.2, *Clerk* produces values for ρ above .85.

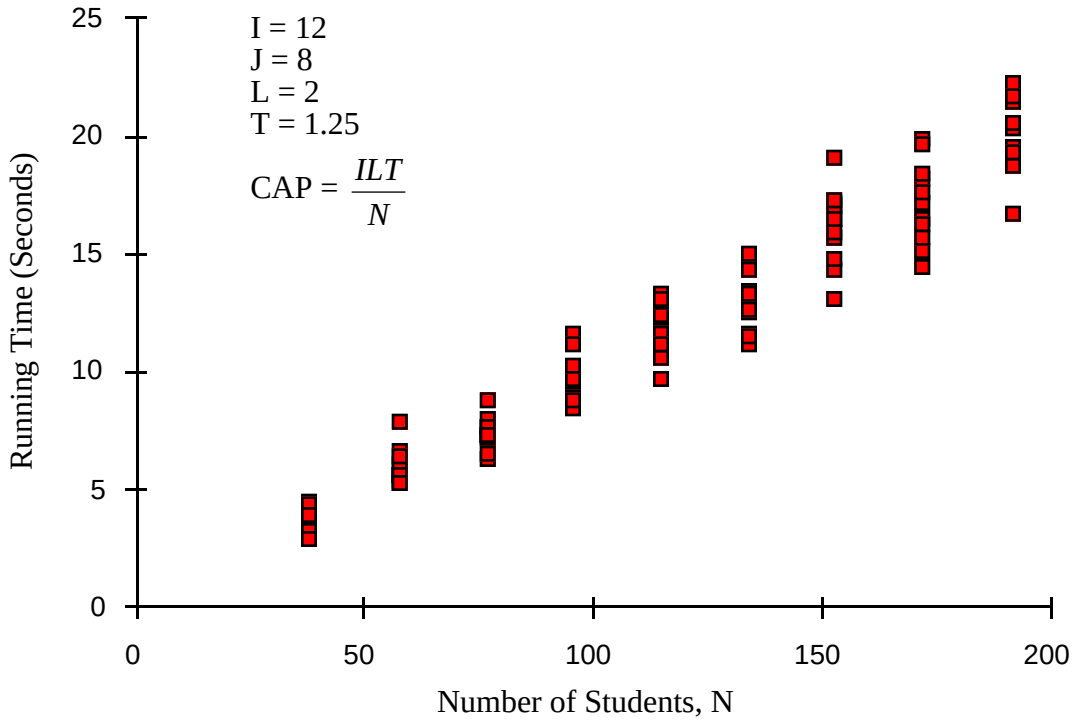
The running time of *Clerk* depends greatly on the number of students:

<u>Experiment D2.2</u>: The running time as a function of N, with constant I,J,L, c_{ijl}.
•Hold I,J,L constant;
• $\forall i,j,l$, c_{ijl} = CAP, where CAP is a constant;
•Run *Clerk*, changing the value of N for each trial.



When N increases, more students need to be scheduled, and rescheduled, which increases the running time. When all other input values remain constant, the tightness also increases as N increases, and this may also effect the running time. So, we should also look at the running time when the tightness stays constant.

Experiment D2.2: The running time as a function of N, with constant I,J,L, and constant tightness T.
•Hold I,J,L constant;
•Run *Clerk*, changing the value of N for each trial, while holding the tightness T constant, where $T = \frac{IL(CAP)}{N}$, by setting $CAP = \frac{ILT}{N}$.



We see from this graph that when T is held constant, the running time seems to be a linear function of N.

In practice, T indicates how many spots are available compared to how many are needed. This will probably remain a consistent value, since no student would be completely denied a program, and hospitals are not going to offer many more opportunities than are needed. So, the running time of *Clerk* can be easily predicted from the number of students to be scheduled.

We have shown above that *Clerk* produces values of ρ that are smaller for tighter schedules. Is this due to a limitation of the algorithm, or is it due to the upper bound U being less reliable for tighter instances? The upper bound U is independent of the tightness T, since the individual optimal schedules ignore the capacities. However, the true optimal schedule for all students must certainly depend on T, since fewer spots are available per student. Thus we can speculate that the smaller values of ρ are due to the fact that U is a less reliable measure of the true optimal schedule for instances when T is close to 1, and not due to a limitation of the algorithm.

Clerk performs very well for scheduling medical students; it gives legal schedules with high happiness values in a reasonable time. The performance of *Clerk* suggests that the Flatten algorithm would perform very well in practice.

Conclusion

Clerkship Scheduling is a four-dimensional weighted scheduling problem. Students(N), times(I), clerkships(J) and locations(L) make up these four dimensions. Different requirements are imposed on each dimension to produce a legal schedule. It is interesting to see how these requirements differ, and how these dimensions depend on one another. A legal schedule must schedule each student to each clerkship. So, a legal schedule must be *complete* in (NxJ); every member of (NxJ) must be covered exactly once. A legal schedule must not schedule any student to more than one time. So, a legal schedule must be *unique* in (NxI); every member of (NxI) may be covered at most once. Finally, a legal schedule must obey the capacity constraints. So, a legal schedule has a *capacity* in (IxJxL); every member of (IxJxL) may be covered at most a fixed amount. These dimensional relationships make Clerkship Scheduling a problem rich in complexity. We have shown that Clerkship Scheduling can solve 3-Dimensional Matching, which is an NP-hard problem.

As a method of approximation, the Flatten algorithm is a useful way to produce legal schedules without enumerating every possibility. It begins by finding each student's individual optimal schedule, thus obeying the requirement that a schedule be *complete* in (NxJ), and *unique* in (NxI). The flatten algorithm continues by changing the schedule to the point where it obeys the capacities of (IxJxL), while maintaining as high a happiness as possible. We have shown an implementation of the Flatten algorithm that performs very well on test data. The Flatten algorihm obeys all the restrictions of Clerkship scheduling, and still produces legal schedules with high happiness values in a reasonable amount of time.

There are many open questions that we did not address. At DMS, spots can have *minimums*, and one of the clerkships also has *prerequisites*; that is, students must complete two specific clerkships before going on this clerkship. In general, how would the complexity of clerkship scheduling be affected by spot minimums and precedence constraints? If I, J, and L are constant, is there a polynomial time algorithm to solve the general problem? We discussed how we may be able to predict whether or not there exists a legal schedule by the tightness of the schedule, but in general, how can we determine if there exists a legal schedule from the input values? For the Flatten algorithm, what is the best way to reschedule dumped students? Additionally, is there a more rigorous analysis of the performance of the Flatten algorithm in terms of the true optimal?

Notes

1) The actual block system used at DMS is slightly different than the one implemented in *Clerk*; Three blocks are used, but the third block only has two clerkships, both of size 2. To implement this change, one would make a few simple modifications to the Match procedure. DMS medical student scheduling also has special cases for *staggered* schedules, where students take a clerkship at the end of the third year.

Bibliography

[1] Garey, Michael R. and David S. Johnson [1979], “Computers and Intractability, a Guide to the Theory of NP-Completeness”, W.H. Freeman and co., New York.

[2] Lawler, E. L.[1976], “Combinatorial Optimization: Networks and Matroids,” Holt, Rinehart and Winston, New York.

[3] Pinedo, Michael [1995], "Scheduling: Theory, Algorithms, and Systems," Prentice Hall, Englewood Cliffs, New Jersey.

[4] Stroustrup, Bjarne [1991], "The C++ Programming Language," AT&T Bell Laboritories, Murray Hill, New Jersey. Addison-Wesley, New York.