The Pennsylvania State University

The Graduate School

The Mary Jean and Frank P. Smeal
College of Business Administration

DEVELOPMENT OF A PROTOTYPE

COURSE SCHEDULING DSS

A Thesis in Business Administration

by

Ole Boysen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December, 2000
Abstract.

This paper presents the development of a decision support system for solving a timetabling problem in the Department of Management Science and Information Systems at the Pennsylvania State University. Currently the problem is solved by hand and this solution procedure is highly time consuming. Although several papers concerning this topic have been published, no solution technique exists for our case, since this kind of problem is extremely variable. In this paper I propose a computerized approach to support the timetabling process and to reduce the time required to solve the problem. This system consists of a graphical user interface (GUI) for the user front-end and an automated solution procedure based on an ant colony optimization algorithm to construct a timetable guided by the scheduling individual’s objectives. Test runs on an actual problem data set indicate, that the solution algorithm is indeed able to solve the instance producing implementable timetables while considering the scheduler’s given objective preferences. The required computational effort is reasonable low.
Table of Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1. INTRODUCTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

| 1.1 Classification of Timetabling Problems | 2 |
| 1.2 Course Scheduling | 3 |

| 2. TIMETABLING AT THE MS&IS DEPARTMENT AT PENN STATE | 9 |

<table>
<thead>
<tr>
<th>3. LITERATURE REVIEW</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

| 3.1 Course Scheduling Models | 21 |
| 3.2 Solution Methods | 22 |

| 3.2.1 Classification | 22 |
| 3.2.2 Ant Systems | 25 |

| 3.2.3 Examples of Applied Educational Timetabling Algorithms | 31 |

| 3.3 User Interaction | 34 |

<table>
<thead>
<tr>
<th>4. THE DECISION SUPPORT SYSTEM</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>37</td>
</tr>
</tbody>
</table>

| 4.1 Database Component | 38 |
| 4.2 Graphical User Interface | 40 |

| 4.2.1 Security | 49 |

| 4.3 Solution Algorithm | 49 |

| 4.3.1 Problem Representation | 49 |
| 4.3.2 Ant System Algorithm | 51 |

| 4.4 Numerical Results | 58 |

<table>
<thead>
<tr>
<th>5. CONCLUSIONS AND RECOMMENDATIONS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>62</td>
</tr>
</tbody>
</table>

| References | 64 |
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database Relationship Diagram</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>The Menu Structure</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>The ‘Solve’ Screen</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>The ‘Course Offering Distribution Sheet’</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>The ‘Teaching Schedule’</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>The ‘Solution Quality Report’</td>
<td>48</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Structure of AS-based algorithms ................................................................. 29
Table 2: Ant System Refinement Options ................................................................. 30
Table 3: Ant Systems for Course Scheduling Pseudo-Code ...................................... 53
Table 4: Results for Different Object Function Weights ............................................. 59
1. Introduction

The problem of assigning a combination of resources and objects to specific time periods without violating some constraints is commonly referred to as the timetabling problem. This paper describes a computerized decision support system for solving a timetabling problem in the Department of Management Science and Information Systems (MS&IS) at the Pennsylvania State University. In particular, this research focuses on the course-timetabling problem in the Department. Currently, a single individual assigns all undergraduate and graduate courses offered by the Department during a semester to timeslots. University, College, and Department scheduling restrictions further complicate the decision making. Her trial-and-error paper-and-pencil method (while yielding acceptable results in general) is highly intuitive, extremely frustrating, and usually takes several weeks to complete every semester. The Department needs a more effective solution technique which addresses the complexities of this timetabling problem directly while enabling the scheduler to apply her expertise to guide the search for good schedules.

While timetabling problems represent a general well-known class of NP-Complete combinatorial optimization problems, every institution’s scheduling problem is unique in terms of its combination of relevant constraints and objectives. As a result, a universally applicable formulation and/or solution methodology for this problem is not offered in the literature, in spite of the numerous publications on the general topic. Prior studies offer insights into general system structure and problem solution strategies, but they offer no definitive framework for this problem instance. Therefore, the focus of this research is to develop an efficient tool to support course scheduling within the MS&IS
Department which reflects the complexities of the problem, enables flexible and intuitive searches for acceptable schedules, and yields good schedules satisfying faculty and student preferences.

1.1 Classification of Timetabling Problems

The problem of assigning a set of resource and object combinations to specific time periods without violating resource limitations and assignment requirements is referred to as the timetabling problem. A general definition of timetabling does not exist, and the term timetabling is frequently used interchangeably with the term scheduling. A subclass of this combinatorial optimization problem is educational timetabling which deals with the assignment of combinations of teachers, students and rooms to time periods without causing conflicts. A conflict arises whenever the timetable assigns a resource (teacher, student, or room) to objects (classes) twice during the same time period. In general, three different types of educational timetabling problems (Bardadym, 1995; and Schaerf, 1995) exist:

- School Timetabling (class-teacher timetabling): This problem seeks a weekly timetable assigning a set of classes and teachers to time periods, so that the timetable schedules no teacher and no class in two teaching units concurrently. Each class has a pre-determined set of pupils who need the same courses and follow the schedule together.
• Examination Timetabling: This scheduling problem (in its easiest case) consists of assigning examinations to time periods so that no student is required to take more than one examination simultaneously. The set of students and the set of examinations of each student are given.

• Course Timetabling: This problem generates a weekly class schedule for a given set of students, who are not grouped and therefore do not require a common set of courses. The set of courses for every student is either known or has to be anticipated prior to timetabling.

No crisp cuts between these problem categories (especially between examination and course timetabling) exist, and some problems cannot be categorized cleanly in one of the three problem categories.

1.2 Course Scheduling

As noted previously, the timetabling problem under study is an example of course timetabling. The following terminology is used to describe the problem. A course is an instruction unit with a specific content and duration. A course can generate one or more classes, also called sections, which are instances of the course with an assigned teacher. A class can be taught in one or more preempted units per week, called sessions or lectures. Each student follows a specific program or curriculum of study which determines his/her requirements for graduation.

The course scheduling problem in its purest form consists of assigning a given set of class-student-room combinations to a restricted set of time periods, so that course and time period restrictions are honored without conflicts and the desired objectives are
achieved. Conflicts arise whenever one of the resources (teacher, room or classes of the same program) is assigned more than once to the same time period. The desired scheduling objectives vary with problem instance. Common objectives include maximizing preferences relative to timeslots (teachers/classes), schedule compactness (teachers) and/or rooms; or minimizing walking distances between lectures for teachers and/or students.

A basic course scheduling problem can be modeled as an integer programming search problem as follows:¹

\[ \sum_{t \in T} x_{it} = k_i \quad \forall i \in I \]  

(1)

\[ \sum_{i \in I} x_{it} \leq l_t \quad \forall t \in T \]  

(2)

\[ \sum_{i \in S_j} x_{it} \leq 1 \quad \forall j \in J, t \in T \]  

(3)

\[ x_{it} \in \{0, 1\} \quad \forall i \in I, t \in T \]  

(4)

where

\[ x_{it} = \begin{cases} 1 & \text{if a session of class } i \text{ is scheduled at period } t, \\ 0 & \text{otherwise} \end{cases} \]

and

\[ i \in I \quad \text{set of classes to be scheduled,} \]

\[ t \in T \quad \text{set of periods available for scheduling,} \]

¹ With reference to de Werra (1985).
kᵢ  number of sessions to be scheduled for class i,

j ∈ J  set of programs of study,

lₜ  maximum number of sessions that can be scheduled at period t,

Sᵢ  set of classes forming program j and thus will have common students.

Constraint set (1) requires all sessions for each class to be scheduled, constraint set (2) restricts the number of sessions scheduled at a single period to an allowable maximum (lₜ), thus taking the number of rooms available into account, and constraint set (3) ensures that no more than one class of a program is offered concurrently. Constraint set (4) restricts all x to be binary variables. In practice, this basic formulation is usually extended with several additional constraint sets to reflect other limiting conditions existing in a real problem instance (for example, class unavailabilities, class preassignments, multiple sections, daily session scheduling maximums, varying length lectures, prior classroom assignments, equipment requirements, and/or walking) and with an objective function which points out the solution quality in terms of the scheduler’s preferences.

The scope (the institutional level conducting the scheduling) and the underlying student registration process can further distinguish pure course scheduling problems. The most common scopes appearing in the literature are central university level (e.g., Paechter, Rankin, Cumming, and Fogarty, 1998), school or college level (e.g., Dinkel, Mote, and Venktaramanan, 1989) and department level (e.g., Rankin, 1995). The scope typically influences the scheduling objective (i.e. preferences) and scheduling methodology opportunities (i.e. if changes to the overall scheduling process can be
enforced). Course registration processes also affect course-scheduling models in that course scheduling may take place before the students have registered for the courses as in *master timetabling* or after registration as in *demand-driven timetabling* (Carter and Laporte, 1997).

In addition to these pure course-scheduling examples, the following scheduling subproblems are often cited (Carter and Laporte, 1997):

- **Faculty timetabling**: This problem assigns teachers to classes while taking faculty members’ preferences into account.

- **Student scheduling/sectioning**: This problem assigns students to courses where the courses have multiple sections so that no conflicts occur, section sizes are balanced and room capacities respected.

- **Classroom assignment**: This problem assigns class sessions to rooms while satisfying size, location and facility restrictions and preferences.

Given the specific constraints, conditions and processes existing at a given educational institution, no universally appropriate model for the course scheduling problem exists. Each scheduling situation represents a unique problem case.

Numerous examples of course scheduling appear in the literature; recent solution frameworks emphasize computerized support systems to provide flexibility and enable more intuitive decision making. For example, Paechter, Rankin, Cumming and Fogarty (1998) present an automated decision support system (DSS) using a genetic algorithm to solve a university-level scheduling problem. Ferland and Fleurant (1994) propose a DSS based on heuristic local search procedure to address a university-level scheduling

Several of these decision support systems have been successfully implemented. However, every one of these systems deals with a specific case of the course scheduling problem, unique to a given institution and not completely representative of the course timetabling problem in the Department of Management Science and Information Systems at the Pennsylvania State University. As a result, this paper draws only general insights from the solution frameworks presented in the literature and explores a case-specific DSS for a Microsoft Windows environment to address the MS&IS Department timetabling problem. As noted earlier, the course timetabling problem is known to be NP-Complete; therefore, heuristic solution procedures are appropriate. This system employs an Ant System algorithm, a metaheuristic, to provide good implementable timetables with reasonable computational effort. Ant System algorithms have been shown to be very
robust in terms of worst-case behavior (e.g. Bullnheimer, Hartl, and Strauss 1997). This robustness and flexibility make an Ant System algorithm particularly appealing given the computational complexity of the underlying multiobjective integer mathematical programming model. Prior course scheduling research also highlights the importance of a flexible transparent user interface to support the decision-making process. As such, this system also includes a graphical user interface to ease data input for the user and enable user-directed searches, provides automatic solution generation, and generates reports highlighting undesirable features of potential schedules. These solutions can be exported to other standard Microsoft Office applications for additional analyses and modifications.

The paper is organized in the following manner. Chapter 2 describes the specific course scheduling problem in the Department of Management Science and Information Systems in detail and presents a mathematical model formulation for this specific problem case. Chapter 3 reviews literature related to problem modeling, solution approach and user interfaces. Chapter 4 presents the decision support system developed and finally, Chapter 5 summarizes this research and suggests future directions for study.
2. Timetabling at the MS&IS Department at Penn State

The particular problem instance under study deals with scheduling all courses offered by the Department of Management Science and Information Systems in the Smeal College of Business Administration at the Pennsylvania State University in a given academic semester. The Department offers graduate and undergraduate courses in the programs of Operations and Information Management, Management Science and Information Systems, Management Information Systems, and Quality and Manufacturing Management. In addition, the Department is responsible for providing several core undergraduate courses (taken by all Smeal College of Business undergraduate students) and core MBA courses (taken by all MBA students).

This scheduling problem consists solely of assigning classes to timeslots (i.e. it is a pure course scheduling problem). The assignment of teachers to classes is completed before class scheduling, and rooms will be assigned after class scheduling by centralized university-wide scheduling personnel. This timetabling is done in a master timetabling fashion before the students actually have requested courses. The scheduling person determines the number of sections of each class by computing the number of students enrolled in each program by semester standing and estimating how many of them are expected to take specific courses in a given semester due to their programs of study. Then every section is assigned to one or more teachers. Furthermore, some courses offered by the Department are parts of programs of other departments and vice versa, and schedule must reflect the needs of these other stakeholder’s needs. To reflect these complexities, the scheduler generates conflict lists containing courses which should not overlap in the schedule, because these courses are expected to be taken simultaneously by
students in particular programs during a given semester. In addition, some classes are divided into lecture and laboratory sessions. Several sections of such a course have a common lecture but separate laboratory sessions, and the laboratory sessions may be required to occur after the common lecture in a week.

With all this information the scheduling person assigns the class-teacher combinations to a given set of time slots. For this purpose the centralized scheduling institution provides time slot sheets, which contain the times of both 50 and 75 minutes time slots available for scheduling. Generally classes have to be held in either three 50-minute sessions on Mondays, Wednesdays and Fridays or in two 75-minute sessions on Tuesdays and Thursdays. However, several exceptions exist—starting times are not absolutely strict, classes do not always fit the given timeslots, and class length may vary from the normal 50 or 75 minutes.

The schedule has to respect an additional set of constraints as summarized below:

1. No classes with the same teacher can occur concurrently.

2. Specific sets of classes cannot occur concurrently because they exist on the conflict list (they belong to the same program and/or students are likely to take these courses in the same semester).

3. A university-mandated spread of the class sessions over the available set of time slots must be satisfied. These restrictions include:

   • No more than 70 percent of the sessions can occupy Monday, Wednesday, Friday slots.

   • No more than 45 percent of the sessions can occupy Tuesday, Thursday slots.
• The ratio of sessions scheduled in a specific set of timeslots (called a row) for either one of the above two weekday allocations must not exceed 15 percent of all sessions.

• The sessions scheduled in early morning timeslots, together with those scheduled for late afternoon timeslots, have to be at least 20 percent of all lectures. However, these restrictions do not apply for classes scheduled after 5:30 p.m., and they have not to be taken into account when making revisions to the schedule in response to student registrations after the schedule is submitted to the university-level scheduler.

In addition to these constraints, the scheduler considers faculty member preferences when creating the schedule. The scheduler tries to incorporate a faculty member’s preferred teaching times and his/her preferred course distribution over the week (back-to-back or spread out).

The scheduling process uses last year’s schedule as a starting point. In fact, the University-level scheduler assumes that the Department’s schedule will stay the same as last year unless the Department scheduler submits changes. As a result, the Department scheduler would prefer to create this year’s schedule by making as few changes as possible to last year’s schedule.

When course scheduling is finished at the department level, the scheduling person for the department submits the final schedule together with room requirements (indicating size and necessary equipment) to the central university-wide scheduling organization. Usually the scheduler must readdress the schedule to resolve room availability conflicts and class conflicts. Also, the number of sections often must be
modified after students have registered in order to accommodate student demand, especially when an over-subscribed course is a required one. Added sections are often scheduled after 5:30 p.m. in order to secure the needed instructional support and classroom facilities.

As suggested above, the nature of this scheduling process is rather unstructured in that a solution’s acceptability or unacceptability is not easily quantifiable. However, a solution’s feasibility or infeasibility relative to course requirements, course conflicts, and university-mandated schedule distributions may be easily ascertained. As such, this problem possesses some structure which can be exploited to find feasible solutions via an algorithm, and the quality of the resulting solutions can be evaluated within a decision support system by the scheduler in an iterative manner until an acceptable schedule is found.

To provide the structure needed for the development of algorithmic support, I depict this problem as an integer programming model as follows.

\[
\text{Minimize } F(x) = \alpha \cdot f_1 + \beta \cdot f_2 + \chi \cdot f_3 + \delta \cdot f_4
\]  

Subject to

Job-Completion:

\[
\sum \sum \sum x_{ijt} \cdot d_{ij} = D_i \quad \forall i \in I
\]
Conflict constraints (Class i must not be held on the same time as class k):

\[
\sum_{k \in R} \sum_{l \in J} \sum_{t \in T} x_{klt} \leq (1 - x_{ijst}) \cdot M \quad \forall \ i \in I, j \in J_i, t \in T_i, s \in S_t
\]  

(3)

Precedence requirements (all sessions of the set \( P_k \) of classes have to be completed before sessions of class \( k \) can start):

a) Check precedence during day:

\[
\sum_{j \in J_i} \sum_{s \in S_i} x_{ijst} \cdot (t + d_j) \leq \sum_{l \in J_k} \sum_{s \in S_i} x_{klst} \cdot t \quad \forall i \in P_k, k \in I, t \in T
\]  

(4)

b) Check precedence over days:

\[
\sum_{t \in T_i} \sum_{s \in S_i} x_{ijst} \cdot t \leq \sum_{t \in T_k} \sum_{s \in S_i} x_{klst} \cdot t \quad \forall k \in I, i \in P_k, j \in J_i, l \in J_k
\]  

(5)

Scheduling scheme (all sessions of a class have to be scheduled at the same timeslot with exactly \( \kappa \) days distance between any two consecutive sessions):\(^2\)

\[
\sum_{j=1}^{|J_i|-1} (x_{ijst} - x_{i(j+1)s(t+\kappa)}) = 0 \quad \forall i \in I, t=1..(|T|\kappa), s \in S_t
\]  

(6)

---

\(^2\) Enforces the sessions to be scheduled in order of indexing and excludes Mo-Fr combinations.
Number of sessions bounds:

a) Minimum number of sessions:

$$\sum_{j \in J, t \in T, s \in S_i} x_{ijst} \geq \lambda_i \quad \forall i \in I$$  \hspace{1cm} (7)

b) Maximum number of sessions:

$$\sum_{j \in J, t \in T, s \in S_i} x_{ijst} \leq \pi_i \quad \forall i \in I$$  \hspace{1cm} (8)

Class i must not be held on same day as class k:

$$\sum_{k \in R, j \in J, s \in S_i} x_{kht} \leq \sum_{j \in J, s \in S_i} (1 - x_{ijst}) \cdot M \quad \forall i \in I, t \in T_i$$  \hspace{1cm} (9)

Course Distribution Requirements (excludes classes after $\varphi$, $|I|$ comprises all lectures incl. prescheduled ones):

(i) On day subset $\theta$ (M+W+F, T+R) scheduled classes $\leq \mu_\theta$ of all scheduled classes:

$$\sum_{i \in I, j \in J, t \in (T \cap \Theta), s \in S_i} x_{ijst} \leq \mu_\theta \cdot |I| \quad \forall \theta \in \Theta$$  \hspace{1cm} (10)
(ii) Totals (row totals) over all days in slot subset (called “row”) u for scheduled classes $\leq \eta$ of all scheduled classes:

$$\sum_{i=1}^{\eta} \sum_{j=1}^{\eta} \sum_{t=1}^{\eta} \sum_{s=1}^{\eta} x_{ijst} \leq \eta \cdot |I| \quad \forall \ u \in U$$  \hspace{1cm} (11)

(iii) Slot subset b totals (early morning and late evening classes) over all days for scheduled classes $\geq \tau$ of all scheduled classes:

$$\sum_{i=1}^{\eta} \sum_{j=1}^{\eta} \sum_{t=1}^{\eta} \sum_{s=1}^{\eta} x_{ijst} \geq \tau \cdot |I| \quad \forall \ b \in B$$  \hspace{1cm} (12)

Binary restrictions:

$$x_{ijst} \in \{0, 1\} \quad \forall \ i \in I, j \in J_i, t \in T_i, s \in S_t$$  \hspace{1cm} (13)

where

$$x_{ijst} = \begin{cases} 1 & \text{if session } j \text{ of class } i \text{ is scheduled in slot } s \text{ on day } t, \\ 0 & \text{otherwise} \end{cases}$$

$|J_i| \in \mathbb{Z}^+$  number of sessions for class $i$

and

$$i \in I \quad \text{set of classes to be scheduled}$$

$$j \in J_i \quad \text{set of sessions of class } i$$

$$t \in T \quad \text{set of days}$$

$$T_i \subseteq T \quad \text{subset of days being available for scheduling class } i$$

$$s \in S \quad \text{set of time slots}$$

$$S_t \subseteq S \quad \text{set of time slots on day } t$$
t_{st} \quad \text{starting time of slot } s \text{ on day } t, \quad t_{st} \geq 0

u \in U \quad \text{set of time slots constituting a “row”, assumed to be at the same to for the purpose of computing the row percentages}

P_i \quad \text{set of direct predecessors of class } i; \text{ classes that have to be finished before the start of class } i

R_i \quad \text{set of classes that cannot be scheduled simultaneously with } i \text{ (conflict list)}

D_i \quad \text{accumulated duration of class } i

\begin{align*}
    d_{ij} &= \frac{D_i}{|J_i|} \quad \forall \ i \in I, \ j \in J_i \quad \text{length of session } j \text{ of lecture } i
\end{align*}

M \quad \text{large number}

\kappa \in \mathbb{Z}^+ \quad \text{number of days between sessions}

\lambda_i \in \mathbb{Z}^+ \quad \text{minimum number of class sessions for class } i

\pi_i \in \mathbb{Z}^+ \quad \text{maximum number of class sessions for class } i

\theta \in \Theta \quad \text{set of days belonging to a sequence}

\mu_0 \quad \text{maximum ratio of classes to be scheduled on days of set } \theta, \quad 0 \leq \mu_0 \leq 1

\varphi \quad \text{time after which scheduled classes do not count for the distribution ratios (5:30pm)}

\eta \quad \text{maximum ratio of classes to be scheduled in each slot subset } u \in U, \quad 0 \leq \eta \leq 1

\tau \quad \text{minimum ratio of classes to be scheduled in slot subset } b \in B \text{ (early morning and late afternoon classes), } 0 \leq \tau \leq 1

\alpha \quad \text{Back-to-back preference weight}

\beta \quad \text{Minimize teaching days preference weight}
Teachers slot preference weight
Minimize schedule changes weight

Constraint set (2) ensures that all classes are scheduled in their entire lengths. Constraint set (3) assures that no conflicting classes occur concurrently. The conflict list $R_i$ for each class includes conflicts due to the program of study, identical teachers, and lecture-laboratory relationships. Precedence requirements between classes are enforced by constraint set (4) for precedence in the case where sessions of both classes occur on the same day and by constraint set (5) in the case where the sessions are held on different days. This relationship is frequently desired for classes with lecture and laboratories. Constraint set (6) respects the scheduling scheme (maintaining a certain distance of days between consecutive sessions of a class and the same timeslot for all its sessions). This means with $\kappa = 2$ that the sessions of a class have to be scheduled in (part-) sequences of either Monday-Wednesday-Friday or Tuesday-Thursday. Constraint sets (7) and (8) restrict the number of sessions for class $i$ to stay within the bounds of $\lambda_i$ and $\pi_i$. Occasionally it is required that a laboratory session is not held at the same day as its corresponding lecture session, and constraint set (9) assures this restriction.

Equations (10) to (12) take care of the class offering distribution requirements, prescribed by the university-wide scheduling institution. Constraint set (10) demands the number of sessions occurring in a subset of days $\theta$ to stay below a certain ration $\mu_\theta$ where $\theta$ is the set \{Monday, Wednesday, Friday\} or \{Tuesday, Thursday\} and $\mu_\theta$ equals 0.7 for $\theta = \{Monday, Wednesday, Friday\}$ and 0.45 for $\theta = \{Tuesday, Thursday\}$. All sessions occurring in a row (i.e. in a subset of slots with a lot of overlapping time) are not allowed
to exceed $\eta$ times the total number of sessions occurring before $\varphi$ by constraint set (11). For example, Row 1 includes timeslot 1 (8:00 a.m. to 8:50 a.m.) and timeslot 13 (8:00 a.m. to 9:15 a.m.) and Row 6 includes timeslots 6 (1:25 p.m. to 2:15 p.m.) and 16 (1:00 p.m. to 2:15 p.m.). Constraint set (12) requires the number of classes scheduled during a subset $b$ of slots to be at least $\tau$ ($\tau = 0.2$) times the total number of sessions before $\varphi$. Under the current scheduling policy, $b$ consists of certain early morning and late afternoon classes and $\varphi$ is 5:30 p.m. Note that $|I|$, the total number of sessions before $\varphi$, is an endogenous algorithm parameter, since it depends on the number of class sessions $|J_i|$ chosen for each class $i$. Finally constraint set (13) restrict $x_{ijst}$ to be a binary variable.

For the sake of reducing notational complexity, the sub-objective functions have been omitted in the mathematical model formulation of the objective function $F(x)$. These sub-objective functions are defined:

$f_1$ Back-to-back preference: adds penalty $\alpha$ for every session of a teacher that does not have another class of that teacher starting or ending within $\gamma$ minutes on the particular day (e.g. one class on Monday of teacher $q$ ends at 8:50 a.m. but the next one starts at 11 a.m. and $\gamma = 60$). The penalty is not given if there is just one class taught by that particular teacher on that particular day.

$f_2$ Minimize teaching days: penalizes each additional day at which the teacher has to teach with $\beta$.

$f_3$ Teachers slot preferences: gives a penalty $\chi$ for every session scheduled at a timeslot not preferred by the teacher.
Minimize schedule changes: penalty $\delta$ is given for every class session of a class, which was scheduled last year, but in a different timeslot or day sequence.

The objectives $f_1$ to $f_4$ are aggregated by summation into a multi-objective function $z$. The weights $\alpha$, $\beta$, $\chi$, $\delta$ are parameters given by the scheduling person and express the relative importance of each of the scheduling sub-objectives in the overall objective function $z$.

After having presented our particular problem under study, similar problems and the methods used to tackle these as they occur in the literature should be examined.
3. Literature Review

The approaches to tackle the scheduling problem vary widely depending on scope, size and constraints of the problem. The focus nowadays lies on decision support since it is generally accepted that not all circumstances can be captured and taken into account by mathematical models and the scheduling person decides which schedule is better on a case-by-case basis. Hence, a great deal of interaction with the user is needed. Furthermore, the acceptance of a system where the scheduling person has the final control over the scheduling process is believed to be much higher than that of a totally automated scheduling system (Foulds and Johnson, 2000). On the other hand, the scheduling problem is so complex that its solution goes beyond human capabilities, which suggests that computers employing algorithmic methods are needed to support the decision process (Ferland and Fleurant, 1994).

This chapter reviews three different levels on which the literature deals with the course scheduling problem: the model level, the solution method level, and the decision support or user interface level. Section 3.1 reviews course scheduling modeling approaches. Section 3.2 discusses combinatorial optimization procedures in general and course scheduling applications in particular. Finally, Section 3.4 presents established course scheduling decision support systems.
3.1 Course Scheduling Models

Models are used to give the particular problem a precise mathematical form. Scheduling models typically consist of a set of hard constraints, which have to be respected to get a feasible timetable, and some soft constraints or objectives, which describe desired but not necessary attributes of a solution. An innumerable number of possible different constraints exist, because every course scheduling problem is unique to some degree. As mentioned above, these constraints can range from forbidding clashes for teachers or students over maximum walking distances between classes to all kinds of requirements concerning the distribution of classes over the week or the day. This variety of problems and the accruing constraints make it impossible to state a general timetabling model formulation comprising all cases.

Various model formulations have been proposed. In his timetabling literature review Bardadym (1995) finds the following representations: reduction to graph coloring models, network flow models, transportation models and integer programming models. For example, De Werra (1985) formulates a course scheduling problem as a graph coloring model. Stallaert (1997) models the problem as a generalized quadratic assignment problem, and Aubin and Ferland (1989) depict the problem as an assignment problem. Dinkel, Mote and Venkataramanan (1989) present a network flow representation while Juretzka, Salewski and Drexl (1998) present a mathematical programming formulation within the framework of partially renewable resources.

However, most reductions of timetabling problems to well-known basic problems (for instance, graph coloring) do not reflect real world problem cases. In these, the simplified models have to be enhanced by additional constraints to capture the real
problem in its full extent. But the extended models cannot be solved by the known specialized solution methods for the simplified standard model representations, thus requiring more general solution approaches.

3.2 Solution Methods

The application of particular solution methods depends on the complexity and scale of the problem. Almost all but very most basic course scheduling problems can be shown to be NP-complete by reduction to an underlying popular problem, which is known to be NP-complete. A further discussion of the algorithmic complexity in timetabling and proof of NP-completeness is given in Bardadym (1995) and Cooper and Kingston (1995).

3.2.1 Classification

Two fundamental different algorithm categories for solving combinatorial optimization problems exist:

(1) Exact algorithms search the whole solution space by enumerating and systematically excluding regions which cannot contain the optimum. In theory, these algorithms find a solution whenever one exists and also the optimal solution in case several solutions to the problem exist. Examples of exact algorithms include complete enumeration and branch and bound algorithms. But, because of the complexity of course scheduling problems, the application of an exact algorithm is rarely appropriate. The

---

3 A problem is considered to be NP-complete, whenever there is no algorithm known, which can solve the problem with at most polynomial effort (Domschke and Drexl, 1991).
computation time explodes with increasing size of the problem and thus only very small problem instances can be solved exactly in an appropriate time frame.

(2) Heuristical algorithms in contrast try to find solutions to the problem quickly by applying objective-directed rules, which are based on certain knowledge or assumptions about the problem. Examples are priority rules and greedy algorithms. Their disadvantage relative to exact algorithms is that the application of a heuristic can never guarantee an optimal solution. In most cases the quality of the solution is unknown. But heuristics are much faster and can provide high quality solutions. Therefore, a heuristic may be the only appropriate alternative to solve large scale, NP-complete combinatorial optimization problems.

Heuristical algorithms may be further differentiated as local search and general search or meta search algorithms. Local search algorithms are simple search algorithms that search in a narrow region. These do not have mechanisms to revive searching once a (local) optimum is reached and therefore likely get stuck at a suboptimal solution. To improve the possibility of a better solution, an algorithm should search the solution space widely without omitting large regions while preventing repeated visits to a sequence of solutions. Algorithms providing these properties are called meta search or general search algorithms (Battiti and Tecchiolli, 1994). The most popular examples of meta search heuristics are tabu search, simulated annealing, genetic algorithms, and ant system, and these techniques are briefly summarized below:
• **Tabu Search**: A move is chosen by a steepest-descent heuristic. Tabu search prevents the revisit of a solution with help of tabu-lists. These lists memorize information about the solution process history. The tabu-lists are used to identify moves which lead back to former solutions and to forbid these moves temporarily.

• **Simulated Annealing**: In every iteration a move is chosen randomly. The move is accepted if it improves the current solution. If the move results in a worse objective function value, it is accepted with a certain probability, which depends on the degree of the deterioration and a so called temperature parameter. This parameter is “cooled down” gradually so that the probability of accepting deteriorating moves is decreasing.

• **Genetic Algorithms**: In contrast to tabu search and simulated annealing, genetic algorithms are evolutionary algorithms, which produce not just one solution per iteration but a couple of solutions (population) in parallel. The next generation of solutions is derived from the current population by hybridization and mutation of selected (good) solutions.

• **Ant System**: This technique also belongs to the class of evolutionary algorithms. Every iteration a population of ants is sent out and every single ant constructs a complete solution. The ants leave trails of pheromone on their path which contain information about the quality of the solution found. This information leads the next generation of ants in a stochastic way and each generation is expanding the existing information.
For a more comprehensive introduction to the first three metaheuristical methods refer to Pirlot (1996). The ant system approach, the heuristic method used for this scheduling problem, is discussed in greater detail in the following section.

3.2.2 Ant Systems

The course scheduling problem is well known to be NP-complete, and therefore, heuristic solution methods are appropriate options to produce good quality solutions with reasonable computational effort. As such, I chose the ant system metaheuristic approach to solve the scheduling problem of the MS&IS Department at Penn State. This technique has been shown, via successful implementations, to be very robust to parameter settings and problem structures. In the following I give a more formal introduction to ant system algorithms.

An ant system algorithm combines a randomized constructive multi-start greedy search procedure with an effective technique for memorizing and using information based on a metaphor of natural ants searching for food. At the beginning, natural ants are completely unguided and wander without focus. During their wanderings the ants leave behind a chemical substance called pheromone. When an ant finds a food source, it takes food and returns to the nest. The ant returning first is the one with the shortest path to a food source. Since ants are attracted to greater amounts of pheromone and this first ant left pheromone on the way to the food source and back (therefore marking it doubly with pheromone) the next ants leaving the nest will be more attracted to this path than the other ones. Following ants will amplify the pheromone trail more and more, and more and more ants will follow this path until the food source is exhausted. The ants will search further for another food source. Since the original path is not used any more, its
Ant systems algorithms imitate the behavior of natural ants finding shortest paths to food sources. Ant systems use artificial ants in a similar manner to solve combinatorial optimization problems, and these artificial ants possess the following characteristics:

1. Ants leave trails containing information about quality and quantity of the food source,
2. Pheromone accumulates faster on shorter paths and
3. Ants choose paths stochastically with preference for stronger pheromone trails.

Colomi, Dorigo and Maniezzo (1991) first introduced the ant system approach for the travelling salesperson problem (TSP), a problem which is very analogous to the natural ant colony’s search for food. The TSP seeks the shortest round trip through a set of cities for a salesperson such that he/she visits each city in the set. The problem is depicted as a connected graph with a set of vertices \( V \) being the cities and a set of edges \( A = \{(i, j) \mid i, j \in V\} \) being the connections between the cities and \( d_{ij} \) being the distance between cities \( i \) and \( j \).

The behavior of the artificial ants is guided by two characteristics: (1) their preference for stronger pheromone trails (i.e. the adaptive behavior) and (2) their greed for shorter paths. I denote the trail intensity (i.e. the pheromone intensity) of edge \((i, j)\) by \( \tau_{ij} \) and the visibility of the distance between nodes \( i \) and \( j \) by \( \eta_{ij} \). Trail intensity and

\[ \text{ Trail intensity and visibility. } \]

\[ \text{ Trail intensity and visibility. } \]

\[ \text{ Trail intensity and visibility. } \]

For a more detailed description of the food searching behavior of ants in nature see Bonabeau and Théraulaz (2000).
visibility are aggregated in a random function to choose the next city to visit in the algorithm. This random function assigns a higher probability $p_{ij}$ to a path from city $i$ to city $j$ when the pheromone intensity of the path is stronger and the visibility of the path is shorter. In contrast to natural ants, the artificial ants can estimate distances and they have a memory $M_k$ to memorize which cities have not been visited yet.

Ant $k$ in city $i$ chooses the next city to be visited $j$ out of the cities in its memory $M_k$ with the following random choice rule (called state transition rule):

$$
p_{ij}^k = \begin{cases} 
\frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{h \in M_k} \tau_{ih}^\alpha \cdot \eta_{ih}^\beta} & \text{if } j \in M_k, \\
0 & \text{otherwise.}
\end{cases}
$$

with

$$0 \leq p_{ij}^k \leq 1$$

probability of ant $k$ in city $i$ choosing city $j$ as the next city to be visited,

$\tau_{ij}$ trail intensity between cities $i$ and $j$,

$\eta_{ij} = \frac{1}{d_{ij}}$ visibility of city $j$ from city $i$ and

$M_k$ set of cities having not been visited yet,

and

$\alpha \geq 0$ controls the relative influence of the trail intensity,

$\beta \geq 0$ controls the relative influence of the visibility.

During each iteration $m$ ants complete a round trip and each of them constructs a complete tour. No information is exchanged during this time (which is known as the self-adaptation phase of an evolutionary algorithm).
New trail intensities are computed at the end of the iteration (when all ants have finished their tours). Therefore, the trail update is delayed (discrete simulation) and globally created in a global trail update. This phase corresponds to the cooperation phase of evolutionary algorithms, which adapts the next generation with help of the information of the last population (Costa and Hertz 1997). The global trail update takes the evaporation \( \rho \) of the old trails during the time of the iteration (the remainder is \((1-\rho) \cdot \tau_{ij}^{t-1}\)) and the new pheromone trails into account. The partial evaporation of trails prevents premature convergence to a particular solution. Since the TSP is a minimization problem, the amount of pheromone left by each ant is the inverse of its solution round trip length multiplied by a scale factor \( Q \). Thus, ants with longer tours \( L_k \) leave less pheromone. The trail change by ant \( k \) is computed by the following formula:

\[
\Delta \tau_{ij}^k = \begin{cases} 
\frac{Q}{L_k} & \text{if ant } k \text{ used edge } (i, j), \\
0 & \text{otherwise.} 
\end{cases}
\]

Consequently the new trail values result from:

\[
\tau_{ij}^t = (1-\rho) \cdot \tau_{ij}^{t-1} + \sum_{k=1}^{m} \Delta \tau_{ij}^k \quad \text{with } m = \text{number of ants.}
\]

The pheromone trails reflect the ants’ collective knowledge about good edges, meaning edges in short tours receive greater amounts of pheromone and will be more likely to be chosen in later iterations. The trail intensity restricts the effective search space more and more in the long run. The following table shows the pseudo-code of ant system-based algorithms.
Several enhancements to the general ant system algorithm exist to cope with implementation issues. For instance, the search occasionally converges too quickly, so that the same solution is constructed over and over again (stagnation), or the search converges too slowly, so that the search takes too long to focus on a particular area and searches little objective-directed space (slow convergence). These problems can be addressed by modifying the algorithm parameters. An increase of $\alpha$ increases the influence of the trail values and accelerates the converging behavior. However, all parameters should be selected so that the algorithm remains robust (the parameters do not have to be changed for every new problem instance). The following table summarizes some of the possible counter measures to common problems mentioned in the literature.

### Table 1: Structure of AS-based algorithms.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize;</td>
<td></td>
</tr>
<tr>
<td>do /* here starts an iteration */</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>position ants on starting points;</td>
<td></td>
</tr>
<tr>
<td>do /* here starts an ant */</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>ant constructs solution using state transition rule;</td>
<td></td>
</tr>
<tr>
<td>save trail update of tour;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>while not all ants have constructed a solution;</td>
<td></td>
</tr>
<tr>
<td>global trial-update;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>while not all iterations finished;</td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Ant System Refinement Options

- Acceleration of converging:
  - Pseudo-random-proportional state transition rule (Dorigo and Gambardella 1996b)
  - Local search/improvement methods (Dorigo and Gambardella 1996a)

- Prevention / repair of stagnation:
  
  Prevention:
  - Local trail-update (Dorigo and Gambardella 1996a)
  - Minimum and maximum trails (Stützle and Hoos 1997a)

  Repair:
  - Trail smoothing / re-initialization (Stützle and Hoos 1997b)

- Acceleration of the method:
  - Candidate lists (Stützle and Hoos 1997a)

In addition, Gambardella, Taillard and Dorigo (1997) introduced the hybrid ant system, modifying the original ant system algorithm as a iterated global search method. The hybrid ant system produces several solutions instead of one per ant and iteration.
3.2.3 Examples of Applied Educational Timetabling Algorithms

A vast number of publications present examples of educational timetabling. As a result, I just present a sample of the more recently published scheduling algorithm approaches, focusing on metaheuristics.

Costa (1994) implements a tabu search algorithm to solve a course scheduling problem. In this application, a move is defined as a reassignment of one class from one period to another. It uses two static tabu lists for different tabu levels—one list bans classes contained in the list from being moved at all and the second just prevents the classes from being moved back to the latest prior assignment. The search diversifies when it does not improve the solution for a certain number of iterations. Diversification occurs by relaxing the objective function weights of some soft constraints for a certain number of iterations. Costa reports that the algorithm delivers applicable good quality solutions.

Dowsland (1998) applies a simulated annealing heuristic to an examination scheduling problem. This approach uses two different neighborhoods applied in different phases of search. The first, move one, is a simple shift move to reassign a single exam to another timeslot. The second, move two, is more complex and uses the concept of Kempe-chains and swaps groups of exams between two timeslots. The annealing uses a geometric cooling schedule.

White and Zang (1998) compare three different approaches for a course scheduling problem—tabu search, constraint logical programming, and a hybrid of tabu search and constraint logical programming. For the tabu search approach, the authors define a neighborhood of two moves: a single shift move which reassigns a single class
to a new timeslot-classroom pair, and a swap where two classes exchange their timeslots and classrooms. The whole timetable is saved in a static tabu list, and diversification and intensification are not applied. Second, the authors formulate the problem as a constraint logic programming problem in PROLOG, depicting the secondary constraints in a constraint hierarchy. The researchers note that the tabu search algorithm delivers better solutions relative to the constraint logic programming approach, but that the constraint logic programming approach is much faster. They obtain the best results by applying the PROLOG program and the tabu search heuristic sequentially. This procedure speeds up the solution process, so that the best solution is found in a fraction of the time the tabu search requires when used alone.

Hertz (1991) implements tabu search techniques for a course scheduling/student sectioning problem by solving the two problem components sequentially with different tabu search algorithms. The course scheduling tabu search considers a neighborhood determined by shift moves that reassign a single class causing a conflict to another feasible period (i.e. no introduced conflicts). This tabu list is static and bans moving a class back to a prior assigned time period for a given number of iterations. The author does not apply diversification or intensification strategies.

Juretzka, Salewski and Drexl (1998) apply simulated annealing and threshold acceptance to a course scheduling problem. For both solution strategies, the neighborhood is defined by shift moves, which assign a class to a different time and another mode (i.e. a room category with rooms of similar capacity and equipment and located within the same area). Temperature and threshold are decreased by a geometric cooling schedule. Computational experiments did not find either of these two algorithms
to be superior to the other. In experiments comparing heuristic performance to conventional solution strategies using a MIP-solver (CPLEX) for small problem instances, both approaches frequently deliver optimal or near-optimal solutions in much shorter times than the conventional optimization approach.

Paechter, Rankin, Cumming and Fogarty (1998) present an evolutionary algorithm for course scheduling and classroom assignment implemented within a decision support system. They choose an indirect representation divided into two parts. Part 1 specifies the order in which the classes will be scheduled sequentially, and Part 2 specifies a number of suggested timeslots for each event. The algorithm constructs a timetable by considering the classes in the order given by a permutation obtained by a multi-point recombination operator. A child inherits its primary suggested timeslot from one parent and its secondary from the other. The algorithm first tries to fit the class into a primary suggested timeslot. If this assignment fails, (perhaps because the assignment would violate some hard constraint), then the algorithm tries a secondary suggested timeslot. If this assignment also fails, a heuristic assigns the class to a timeslot (where the timeslot might contain unscheduled classes). Lamarckian write-back and three different mutation operators refine the algorithm.

All of the implemented approaches are reported to deliver solutions of user satisfying quality. Unfortunately, a comparison of solution quality between the solution procedures is impossible, since the studied problems differ and the solution procedures are applied more or less problem specific. Furthermore, the justification for using a particular heuristical solution approach, without implementing and studying all of them for the same problem, is an area of science that has not progressed very far yet.
Another issue to consider while making the decision for a particular solution approach and designing it, is the kind of interaction between the solution procedure and the user, that is required.

### 3.3 User Interaction

The degree of interaction in course scheduling systems found in the literature varies greatly. Fully automated timetabling systems allow a user to input data and then the system generates a complete solution. Highly interactive systems enable a user to define parameters to fit the model to his/her particular problem before an algorithm solves the scheduling problem. The resulting timetable then can be modified and the system supports such modifications by other optimization algorithms suggesting alternatives for desired changes. The degree of scheduling support varies from simple visual help to depict resource availability as in Mathaisel and Comm (1991) over completely automated solution to interactive support with algorithm-based suggestions for every single decision. As stated above, the variety of constraints and preferences in course timetabling problems cannot be captured in mathematical models completely and unstructured case-by-case decisions are involved. Therefore, the system must supply the scheduling user with intense system interaction opportunities. To equip systems with user-friendly input, output and interaction opportunities, which are easy enough to use without a great deal of training, the use of modern graphical user interface environments (like the Microsoft Windows environment) becomes mandatory.

The review of literature can be easily restricted to recent publications, since the opportunities for graphical user interface programming have improved a great deal during the last few years. Carter and Laporte (1997) present a comprehensive survey of recently
published course scheduling systems that have been implemented and tested with actual
data in real problem instances. First, Aubin and Ferland (1989) describe an implemented
system addressing a university-level demand-driven problem with teacher-class-room
assignments and student sectioning. They use a heuristic exchange procedure to solve the
problem. Second, Rankin (1995) presents a system that generates timetables
automatically and provides visual support detailing conflicts and opportunities for
adjusting the schedule. The system uses a genetic algorithm to solve the problem.
Paechter, Rankin, Cumming and Fogarty (1998) revise the original Rankin system (1995)
to improve the solution procedure by introducing a graphical interface where the user can
better direct the search by changing relative objective weights in a multiobjective
function during the search process. Finally, the scheduling system of Sampson, Freeland
and Weiss (1995) deals with a school level demand-driven course scheduling and student
sectioning problem where they take teacher preferences into account. They employ a
totally automated local search heuristic to generate timetables.

More recent publications include a system called SAPHIR by Ferland and
Fleurent (1994). This system can generate a timetable from scratch but it also supports
manual improvements to a suggested solution in a highly interactive manner. It attacks
the course scheduling problem as well as the student sectioning problem on the university
level in a demand-driven fashion and it is a revised version of the Aubin and Ferland
(1989) system using a similar heuristic local search algorithm. Stallaert (1997) presents a
department-level master timetabling system. A heuristic algorithm generates a timetable
automatically, and the system prints reports of conflicts. Foulds and Johnson (2000)
propose another master timetabling support system called SlotManager. SlotManager, a
school level system at a New Zealand university, is highly interactive in supporting user changes but does not offer automated timetable generation.

As suggested by the above discussion, each of these timetabling problems is unique, reflecting the unique characteristics, policies, and priorities of scheduling at the institution under study. Furthermore, every single one of the reviewed scheduling problems is different than the timetabling problem in the Management Science and Information Systems Department at Pennsylvania State University. The cited systems show, that a successful implementation would require to allow the user to depict preferences for multiple desired objectives, to provide an automated generated timetable considering these preferences and to support by-hand-modification of the timetable via bookkeeping functions, that point out resource availability and statistics, and algorithms, which suggest scheduling alternatives. All but the by-hand-modification support have been integrated into the developed course scheduling DSS as described in the following chapter.
4. The Decision Support System

The decision support system developed to address the course scheduling problem in the Management Science and Information Systems Department consists of three main components: the graphical user interface, the database and the solution procedure. The graphical user interface (GUI) has been programmed in Microsoft Visual Basic 6, the database uses the Microsoft Access Jet Database Engine 4 and the solution procedure has been developed in Microsoft Visual C++ 6. The user inputs data concerning classes, teachers and timeslots into the GUI which organizes and stores that data in the database component. The GUI prepares the data and transfers it to the solution algorithm which solves the problem taking into account problem and algorithm parameters provided by the user. The algorithm employs an Ant System metaheuristic to solve the problem and returns the best solution found in the timeframe prescribed by the user. The GUI evaluates the solution and prints different representations of the schedule and a solution quality report, which exhibits potential problems in the solution generated.

In this chapter, I first explain the database design in Section 4.1. Then Section 4.2 describes the graphical user interface component and Section 4.3 details the solution algorithm. Last, Section 4.4 evaluates algorithm performance and usability by applying it to real problem instances.
4.1 Database Component

The database is designed in Microsoft Access 97 and employs the Microsoft Jet Database Engine 4. It includes nine tables: Courses, Course Conflicts, Teachers, Teachers Preferences, Timeslots, Early and Late Timeslots, Rows, Classes and Class Distribution requirements. The database performs bookkeeping functions in this system, rather than providing the user with real database functionality. The main advantage of using a database instead of a collection of flat files is that the database format offers easy access to data integrity features. Since almost all data is related to each other, the programming effort to check the integrity of the data over all files without using a database format would require great effort.

All data inputted through the GUI is stored in the database, and each semester a new database instance has to be created. To start the scheduling process for a new semester, the user loads the database from last year and saves it under a new filename. This retains data, that rarely changes, in the following called permanent data (i.e. timeslots, courses, teachers). The user modifies mainly the frequently changing data, which is semester dependent (i.e. classes). The relationship diagram of the database is shown below in Figure 1.
Figure 1: Database Relationship Diagram
4.2 Graphical User Interface

The GUI includes thirteen input screens and a help system. The main screen shows the main menu bar and serves as a container for the other screens. The following figure shows the complete menu structure.

The top row of the above figure shows the main menu of the GUI (e.g. menu bar). Each main menu point is connected to its submenu points (drop-down menu) by arrows, corresponding to identically titled screens. The screens in the second level of indentation are accessible directly from their parent screens. The zero-level main menu differentiates between permanent and changing data. Changing data is all data likely to require changes every semester (e.g. the classes offered and the instructors assigned to teach these classes,
special circumstances requiring classes to be prescheduled at a certain times, and the schedule data for the classes from last year). In contrast permanent data is all data likely to stay almost the same from semester to semester with little change (e.g. adding a new instructor or a new course).

The interface should be rather self-explanatory since it employs the familiar Microsoft Windows environment and navigation is completely mouse driven. Therefore explanations of every single screen are omitted. Instead I discuss how some important scheduling circumstances can be realized within the presented environment.

- **Conflicts:** The system automatically recognizes infeasible solutions due to instructor conflicts. To prevent course conflicts, the user can enter courses that should not be offered at the same time. For course containing lectures and a laboratory section, the lecture section always inherits the conflict list of the course from which it is derived. Laboratory sections can be selected to inherit the course’s underlying course conflict list. However, if several laboratory sections are planned, it is probably not necessary to recognize the course conflict list explicitly, since students do not take all of the conflicting classes simultaneously and numerous laboratory scheduling options exist. Furthermore, omitting the conflict list for lab sections gives the solution procedure more freedom and can result in better solutions.

- **Laboratory sections:** Within the DSS, a lecture class and its associated laboratory are treated as two separate classes possessing a parent-child relationship. Since the parent lecture class can have multiple laboratory classes associated, the lecture class is not identified by with a unique class code number. The user inputs a zero as the lecture class number. The user must also input session bounds for the lecture sessions
and the system derives session bounds for the laboratory class as the difference between the duration specified for the underlying course and the parent lecture class. Likewise, the minimum and maximum sessions of a lecture and its laboratory are separately noted. If a course has just one lab session and this lab is taught by the same teacher as the parent lecture and should be taught in the same timeslot as the parent lecture, the course can be entered as a single course without differentiating between its lecture and laboratory sessions. The solution procedure automatically prevents conflicts between lecture classes and their associated laboratory classes.

- **Undefined teachers:** If the teacher for a class has not yet been determined, the instructor’s name can be unspecified. In this case the algorithm assumes that the class will not have a conflict due to its teacher. But if a dummy name for the teacher is entered, the algorithm treats the dummy name as a real name and attempts to resolve instructor conflicts for all classes with the same teacher. This feature is most helpful in the case of several laboratory sections. In this case the same instructor name (real or dummy) could be entered for all sections to have the algorithm schedule all sections at different times. On the other hand, if it is not necessary to have all sections at different times, a different instructor name (real and/or dummy) for every section can be entered so that the algorithm has more freedom for scheduling.

- **Session duration:** Sessions of 50 or 75 minute duration are required to fit into the timeslots given by the general University scheduling policies. However, these durations do not reflect the desired scheduling for some graduate classes within the Department, for faculty prefer teaching some graduate classes in a single weekly 150 minute session. The solution procedure incorporated into the DSS provides this
desired flexibility. The DSS focuses only on the beginning time of the timeslot, not the timeslot end. This approach means, for example, that a 150-minute class taught in one session of 150 minutes could be assigned to any timeslot (50 or 75 minutes long). In contrast a 50-minute session of a three-session class can only be assigned to a timeslot with a length of 50 minutes, not to a 75-minute timeslot. As a result, the system cannot reproduce those manually-created timetables using 75-minute slots for 50-minute sessions and vice versa.

- **Objective weights:**

  ![Figure 3: The ‘Solve’ Screen](image)

  Figure 3 shows the *Solve* screen of the DSS. The scheduler uses the four sliders to indicate the weights $\alpha$, $\beta$, $\chi$, and $\delta$ applied to each of the components of the overall
objective function. The value entered is the penalty given for every violation of the particular component of the overall objective function. Therefore, these parameter values do not just represent the weights relative to the other scheduling objectives, but they also directly represent the scheduling objective’s weight within the overall objective function as defined in Chapter 2.

The objective function contains penalties for hard constraint violations and for soft constraint violations. As such, it is possible for the user to choose scheduling objective weights that restrict the objective solution space so much that a feasible solution (i.e. a solution respecting all constraints regardless of the scheduling objectives) cannot be found. In this case, the weights should be reduced to allow the algorithm to focus more on the search for a feasible solution.

- **Solution approach:** The solution algorithm is stochastic, producing a different solution (almost) every time it is started. Multiple timetable suggestions may be obtained by simply re-starting the algorithm.

The user-specified search duration is critical. As a rule of thumb the solution time should not be reduced below twenty seconds, depending on the computer used and the scale and complexity of the problem instance. However, I do suggest using short algorithm runs of about ten seconds each to verify that the specified objective weights guide the algorithm in the desired direction and that all required restrictions for the classes have been entered. Afterwards, the scheduler can specify longer runs to get final timetable suggestions.
• **Solution reports:** The automatically generated timetable is represented in two different reports—the class offering distribution sheet and the teaching schedule. In addition, the system provides a report summarizing the quality of the resulting schedule. All of these reports can be exported as text files for further use in other standard office applications.
Figure 4 provides an example of the class offering distribution sheet, an output report with the same structure as a paper report historically distributed by the university-level scheduling function at the Pennsylvania State University. This form gives a quick overview of all classes organized by the row and timeslot in which they are scheduled. The right border shows for each row the total number of class sessions scheduled in the timeslots of that row. The lower border shows the total number of sessions scheduled on the particular day before the point specified for excluding classes from distribution requirement calculations (normally 5:30 p.m.). The right corner shows the overall session total for all classes scheduled before this point in time. These sums are used to compute the distribution requirement measures.

**Figure 4: The ‘Course Offering Distribution Sheet’**
Figure 5 provides an example of the teaching schedule, the second timetable representation provided by the DSS and similar to the paper schedule currently distributed to faculty member. This screen shows all classes organized by course, section number and code number.

![Figure 5: The 'Teaching Schedule'](image-url)
Finally, Figure 6 presents the *Solution Quality Report*. This report highlights potential problems in the generated timetable. It lists all constraints, that are violated. Also, it lists undesired objective violations for not respected back-to-back preferences, not respected timeslot preferences and changes from last year's schedule. Not listed are not respected minimize teaching days objectives; determining the minimum number of days required is a difficult task itself because of prescheduling, possible day codings for classes, and resulting possible day combinations.

**Figure 6: The ‘Solution Quality Report’**

![Solution Quality Report](image)
4.2.1 Security

The program never opens the chosen database itself but creates a temporary copy of the selected database ‘temp.mdb’. Therefore, undesired changes to the data can be undone by simply reopening the particular database. The database is reset to the status where it has been saved last. In case the system crashes, the most recent version of the database can be found in the file ‘temp.mdb’ within the application directory. To save the database status at the time of the error, the user must copy ‘temp.mdb’ to another filename before restarting the Course Scheduling application. Of course, the user should save frequently to prevent data losses. The older version of the saved database is renamed with the same file name but includes the file extension ‘.bak’.

4.3 Solution Algorithm

As mentioned previously, the system employs an Ant System metaheuristic to solve the problem. In this section, I first explain how the problem is represented for algorithm implementation, and then I explain the algorithm itself.

4.3.1 Problem Representation

The problem is implemented as an assignment problem. As such, the task is to assign a given class-teacher combination to a timeslot-code combination respecting the constraints specified in the integer programming model. A code is a bit representation of an available day-sequence (e.g. Monday-Wednesday-Friday has code 10101 binary equal to 21 decimal). The search space of the procedure is narrowed compared to the integer programming model explained above, since the search procedure enforces some of the
hard constraints implicitly. All classes are scheduled and the duration of sessions is computed after choosing the number of sessions for the class so that job-completion (constraint set 2) is assured. The code is chosen from the set of codes being feasible for the particular class in terms of the minimum and maximum session bounds (constraint sets (7) and (8)). By choosing a code the day-distance constraints (constraint set 6) is respected implicitly since the list of codes from which to choose contains just codes respecting these constraints. All other constraint sets ((3), (5), (5), (9), (10), (11) and (12)) from the integer programming model are implemented as soft constraints and can therefore be violated during the search. Violations of these constraints are penalized and aggregated together with the desired weights in the overall objective function. However, the penalty for violating a soft constraint is very high so that a schedule with one soft constraint violation can never be evaluated better than a schedule without any soft constraint violations. The objective function is designed as a minimization one, thus penalties add positive values and a higher objective function value indicates a worse solution.
4.3.2 Ant System Algorithm

This section explains the choice and implementation of appropriate algorithm components. A lot of effort can be spent choosing components of algorithms, since extensive experiments and statistical analysis can reveal advantages and disadvantages of these choices. It is incorrect to assume that experiments with other optimization problems would yield the same results or that these results would confirm an algorithm’s performance in other problem instances with different structures. However, the narrow timeframe of this project and the absence of problem instances precludes such experimentation. Therefore, my choices of components and parameter values rely on results and conclusions taken from literature and from my own experience and intuition.

In this section, I first briefly outline the solution procedure. I then provide greater detail about the single components of the algorithm implemented. Table 2 describes other possible component choices.
**Solution Procedure**

In brief, the solution procedure is as follows:

Every iteration includes a population of k ants, of which each single ant produces a complete solution. For each ant, the algorithm first produces an order for the classes in which they are to be scheduled. A randomized priority rule determines this order, which gives a class a higher priority if it has more conflicts in its conflict list. The list is processed successively and always the class on top of the list is scheduled next. Then the algorithm chooses a timeslot/code-combination for that class by applying a so-called pseudo-random-proportional state transition rule. The class is scheduled in the chosen timeslot/code combination and removed from the ordered list. The algorithm begins processing the next class at the top of the list in a similar manner and so forth. After scheduling all classes, a greedy heuristic is applied to improve the obtained solution and the algorithm puts the next ant on start. When all ants have produced a schedule, the trails are updated, and trail loses some of its pheromone due to evaporation. Then all trails used in the best solution found so far are reinforced. If no new best solution is found for a specified number of iterations, all trails are reinitialized, thereby deleting all trail information. This deletion does not mean a complete restart of the algorithm since the algorithm still uses the best solution found so far for updating the trails. Then the next iteration starts and the algorithm continues in the same manner. The algorithm terminates when the specified number of iterations has been reached or a specified time limit has been exceeded. Table 3 presents this high-level Ant System algorithm in pseudo-code.
Table 3: Ant Systems for Course Scheduling Pseudo-Code

\[ F(x^*) = \infty; \] // \( F(x^*) \): objective function value of best solution \( x^* \)
\[ x = \emptyset; \] // \( x \): solution
Preprocessing(x); // evaluate and schedule prescheduled classes
\[ x_1 = x_2 = \ldots = x_{\#ANTS} = x; \]
lastImprove = 0;
for (Iter = 1; Iter \leq \#ITER; ++Iter)
{
    \[ F(x^{Iter}) = \infty; \]
    Init_Trails(\Delta Trail);
    for (ant = 1; ant \leq \#ANTS; ++ant)
    {
        \[ x = x_{\text{ant}}; \]
        ListI = random[I]; // create random order class list
        while (|ListI| > 0)
        {
            i = top[ListI]; // get top most class from list
            ListI = ListI \{i\};
            (s, c) = ChooseAssignment(i); // choose timeslot \( s \) and code \( c \)
            \[ x_{\text{ant}} = x_{\text{ant}} \oplus m(i, s, c); \]
        }
        Improve(x_{\text{ant}}); // improvement heuristic
        if (F(x_{\text{ant}}) < F(x^{Iter})) // new iterationbest solution?
            \[ x^{Iter} = x_{\text{ant}}; \]
    }
    if (F(x^{Iter}) < F(x')) // new best solution?
    {
        lastImprove = Iter;
        x' = x^{Iter};
    }
    if (lastImprove + maxNonImprove < Iter) // stagnation assumed
        TrailReinit();
    Else
    {
        Update_\Delta Trail(x^{Iter}, F(x^{Iter}));
        Update_\Delta Trail(x_{\text{best}}, F(x_{\text{best}}));
        GlobalTrailUpdate(\Delta Trail); // reinforce trails
    }
}
\textit{Trails}

The first decision made during the implementation of an ant system algorithm concerns the trails. Specifically, what is connected by trails? During the construction of a timetable, each move consists of two successive decisions: (1) Which class should be scheduled next and (2) to which timeslot/code combination should it be assigned? Since the second question has apparently the greater impact on the solution quality, I decided to draw pheromone trails from each class to all timeslot/code combinations. But also alternative (1) would make sense, since a simple greedy algorithm is able to produce an optimal solution when the order of classes being scheduled is right.

Thus the representation chosen is: For each class \(i\) there is one trail \(\text{Trail}(i, s, c)\) leading to each combination of timeslot \(s\) and code \(c\). For instance, a typical class of 150 minutes has a minimum number of sessions equal to two and maximum number of sessions equal to three and four possible codes: MWF for an assignment with 3 sessions, and MW, TR, and WF for two sessions (note that MF sequences are not desired). Since the actual timeslots sheet contains 20 timeslots, there are \(4 \times 20 = 80\) trails for each class or 80 different assignments possible.

\textit{State Transition Rule}

The state transition rule is used to determine the assignment of a class (i.e. the timeslot/code combination to which it is assigned). The pseudo-random proportional rule applied selected for application to this problem has been evaluated to be superior to the original random-proportional state transition rule in comparative studies presented in the literature (e.g. Gambardella and Dorigo 1995). This rule chooses the best move available
in terms of the weighted combination of trail and visibility with probability $q_0$ or to make a randomized choice with probability $1-q_0$. The randomized choice assigns a timeslot/code combination a higher priority given higher combined values of trail and visibility. Thus, the procedure to choose an assignment for class $i$ is:

1) Choose a uniformly distributed number $q$ between 0 and 1.

2a) If $q < q_0$, choose the combination $(s, c)$ so that:

$$\max_{\forall s \in S_c, c \in C_i} \left( \text{Trail}(i,s,c)^\alpha \cdot \left( \frac{1}{F(x \oplus m(i,s,c))} \right)^\beta \right),$$

2b) otherwise choose a timeslot/code combination $(s, c)$ using random function:

$$p(i,s,c) = \frac{\text{Trail}(i,s,c)^\alpha \cdot \left( \frac{1}{F(x \oplus m(i,s,c))} \right)^\beta}{\sum_{\forall s \in S_c, c \in C_i} \left( \text{Trail}(i,s,c)^\alpha \cdot \left( \frac{1}{F(x \oplus m(i,s,c))} \right)^\beta \right)}.$$

Note that timeslot $s$ and code $c$ are chosen from the set of feasible combinations in that $c \in C_i$ is feasible when the number of days of $c$ is within the boundaries of the specified minimum and maximum sessions for class $I$, and $s \in S_c$ is feasible when timeslot $s$ is offered on every single day of code $c$. The visibility uses the inverse of the objective function value $F(x)$ since this timetabling problem is a minimization problem.

To prevent violations of precedence constraints when determining the best move, the following tie-breaking rule is applied:

In case move $m$ gets the same evaluation as the actual best move $m^*$ for class $i$, it becomes the new best move, if class $i$ has a predecessor (successor) and the first (last) day of move $m$ is later (earlier) than the first (last) day of move $m^*$. 
The bias towards the best move is supposed to provide a more strictly aimed search, and therefore, faster convergence toward good solutions.

**Global Trail-Update**

The global trail-update takes into consideration the best solution found so far. This update applies the following formulas:

\[
\Delta \text{Trail}(i,s,c) = \begin{cases} 
\frac{Q}{F(x^*)} & \text{if } (i,s,c) \in \text{actual best solution } x^*, \\
0 & \text{otherwise.}
\end{cases}
\]

\[
\text{Trail}(i,s,c) = (1 - \rho) \cdot \text{Trail}(i,s,c) + \Delta \text{Trail}(i,s,c)
\]

with

\[
Q = F(x_1^*) \text{ denoting quantity trail, and } x_1^* \text{ is the best solution found in iteration 1.}
\]

In order to restrict the trail influence, the trail values are bounded by \( \tau_{\min} \) and \( \tau_{\max} \). These values are chosen by:

\[
\tau_{\min} = 1
\]

\[
\tau_{\max} = \tau_{\min} + \frac{Q \cdot \text{maxTrailFactor}}{F(x^*)}
\]

where \( \text{maxTrailFactor} \) is a fixed parameter.

Note that \( \tau_{\max} \) is increasing dynamically with the change of \( F(x^*) \). This increase allows the trail information to get more influence during later stages of search (for the concept of minimum and maximum trail values, the reader should refer to Stützle and Hoos (1997b)).
**Trail Re-initialization**

To prevent the solution process from becoming completely stuck, the search is diversified from time to time. An iteration counter that is reset whenever a new run-best solution (i.e. best solution since last trail re-initialization) is found determines the moment of diversification. If the algorithm does not find a new run-best solution before the counter exceeds a specific threshold, the diversification mechanism is triggered and the run-best solution is set to infinity. In order to diversify the search, all trails are then re-initialized to $\tau_{\text{min}}$. Note that this re-initialization does not mean a complete restart of the algorithm, because the trail update still uses the best solution found so far.

**Local Improvement Heuristic**

The algorithm applies a steepest-descent heuristic to the solution of every ant to improve the ant’s solution. As such, the algorithm evaluates all possible reassignments of single classes to combinations of timeslots and codes (shift moves) to determine best fit. If a reassignment exists that improves the actual solution, the reassignment is carried out otherwise the improvement procedure is exited. The process is repeated a specified number of times or until an iteration does not produce an improved solution.

The usability of the algorithm designed has to be evaluated by test runs. It has to be proved, that the algorithm avoids conflicts and considers the objective preferences given by the scheduling person, while keeping the computational effort in reasonable limits.
4.4 Numerical Results

I evaluate the performance of the algorithm by applying it to the only problem instance available hitherto, the scheduling problem of Fall Semester 1999 at the Department of Management Science and Information Systems at the Penn State University. This data set contains 34 instructors, 23 timeslots, and 71 classes, from which 19 are prescheduled. To evaluate the procedure’s ability to distinguish between scheduling objective preferences, the basic problem instance is modified with different settings for the scheduling objective function weights for the four preference criteria (Back-to-Back ($\alpha$, BtB), min.Days ($\beta$, minD), TeachUnavail ($\chi$, TUnav), and SchedChg ($\delta$, SChg)) as shown Table 4. This table shows the given objective function weights on the left side and the resulting number of soft constraint violations on the right side, each obtained by algorithm runs of 20 seconds on an AMD Athlon 700 MHz processor with 64 megabytes of memory. Please note that the first four rows depict results when each scheduling objective is considered independently. The remaining twenty weight combinations shown in this table were randomly generated.
Table 4: Results for Different Object Function Weights

<table>
<thead>
<tr>
<th>Objective Function Weights</th>
<th>Number of Violations</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BtB</td>
<td>MinD</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>29</td>
<td>97</td>
</tr>
<tr>
<td>42</td>
<td>22</td>
<td>98</td>
</tr>
<tr>
<td>45</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>91</td>
<td>12</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>91</td>
</tr>
<tr>
<td>57</td>
<td>35</td>
<td>65</td>
</tr>
<tr>
<td>94</td>
<td>99</td>
<td>33</td>
</tr>
<tr>
<td>74</td>
<td>65</td>
<td>82</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>56</td>
</tr>
<tr>
<td>68</td>
<td>93</td>
<td>58</td>
</tr>
<tr>
<td>95</td>
<td>31</td>
<td>56</td>
</tr>
<tr>
<td>43</td>
<td>88</td>
<td>14</td>
</tr>
<tr>
<td>22</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>46</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>29</td>
<td>69</td>
</tr>
<tr>
<td>98</td>
<td>36</td>
<td>3</td>
</tr>
<tr>
<td>91</td>
<td>96</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>76</td>
</tr>
<tr>
<td>17</td>
<td>73</td>
<td>71</td>
</tr>
</tbody>
</table>

All of the problem instances have been solved to feasibility (e.g. the resulting timetable contains no time / resource conflicts and therefore is implementable). Note that the objective for teacher timeslot preferences UnAva is always zero, since the data set
includes the timeslot preferences for only one instructor. The other results show that the algorithm is guided by the given objective weights. This observation can be further verified by hypothesis testing using the Pearson correlation. As such, we test

$$H_0: \text{the values are not correlated}$$

vs.

$$H_1: \text{the values are correlated}.$$ 

The test results for the three objective weights involved are:

- BtB and BtB violations: correlation = -0.73, p-value = 0.000
- minD and minD violations: correlation = -0.689, p-value = 0.000
- SChg and SChg violations: correlation = -0.560, p-value = 0.004

Thus, the null-hypothesis can be rejected in each case indicating that a strong correlation exists between the value of the weight and the number of violations of the corresponding objective.
Any evaluation of the algorithm’s usability is complicated by the fact that all of the objectives considered when creating a schedule for the MS&IS Department are not completely structured and quantifiable, and they probably never will be. Thus, a comparison between the timetables generated by hand and by the algorithm can be measured only by the objectives integrated in the objective function. However, the scheduling person might prefer another solution due to these non-quantifiable criteria, which are often decided on case-by-case basis. But as stated before, a completely automated solution is not the goal of this paper. Our goal is to provide automatically generated feasible timetables that could serve as a starting point for user adjustments and speed up the overall solution process.
5. Conclusions and Recommendations

This paper presents a novel scheduling decision support system for the course scheduling problem in the Department of Management Science and Information Systems at the Penn State University. In particular a graphical user interface (GUI) has been developed, which enables easy input of data and user preferences. Furthermore, the system includes an ant system algorithm to solve the entered problem instances to feasibility while optimizing for several scheduling objectives according to the user’s preferences. The user can review the schedules as presented in three different reports, adjust the input data and preferences, and resolve the problem as needed. Then the solution can be exported for further modifications with standard office software.

As stated in the introduction, solution to the course scheduling problem cannot be completely automated, but rather it must be solved in interaction with the scheduling individual. Therefore, I envision highly interactive modification procedures that could be integrated in this GUI to suggest possible reassignments of classes as guided by the user. In addition, screen design for such modification procedures remains an important question — in other words, the screen should give a useful overview of the entire timetable and point out resource information and reassignment suggestions in a transparent and helpful way. Moreover, data input via the world-wide web should be explored. For example, the edit sheets for the teachers’ timeslot preferences could be made accessible via the world-wide web enabling direct instructor data input, thereby decreasing the workload of the scheduling person and increasing the system’s adaptability to teachers preferences and their satisfaction (more complex preference structures could be applied).
Several refinements are possible with respect to the algorithm. First, experiments to fine-tune algorithm parameter choices should be conducted as soon as more problem instance data becomes available. Additional effort could be spent to optimize the algorithm code to gain solution speed. However, additional speed is not needed at this point in time, because the algorithm located feasible solutions in mere fractions of seconds in the experiments reported in this study. Second, the algorithm should be better adapted to the underlying characteristics of the specific problem to improve solution quality and speed. It is thought that algorithm-specific improvements (like, for instance, the use of a neighborhood search instead of a constructive solution generation) may speed up the search a great deal. Problem-specific refinements would enable us to get closer to the preferences of the scheduling person. These refinements require further structuring of the problem and could, for example, be concerned with improved tie-breaking rules to get a better spread of classes over the week.

Last, a decentralized university-wide scheduling system could be examined. The challenge would be to find a central negotiation mechanism, which maintains the individual schedulers’ preferences while seeking overall optimization with respect to university preferences (i.e. room utilization or preferred core times). This would have to be accomplished without increasing organizational and communicational effort, especially between the departments. The departments’ degree of autonomy in scheduling should be retained.
References


