Stundenplan-Erstellung mit evolutionären Algorithmen - ein Softwareprojekt

Schriftliche Hausarbeit im Rahmen der ersten Staatsprüfung für das Lehramt für die Fächer Informatik und Mathematik

Dem Landesprüfungsamt Brandenburg für Erste und Zweite Staatsprüfungen für Lehrämter an Schulen

vorgelegt von : Jan Weingärtner
Potsdam, November 1999

Themensteller: Prof. Dr. Andreas Schwill
Universität Potsdam
Institut für Informatik/ Didaktik der Infomatik

1. Einführung

In vielen Bereichen des täglichen Lebens steht der Mensch vor der Aufgabe, mit möglichst geringem Aufwand möglichst hohen Nutzen zu erzielen. Er wird also mit Problemen konfrontiert, für die er die bestmöglichen Lösungen sucht. Als besonders schwierig stellt sich immer wieder die kostengünstigste Verteilung von Ressourcen im Bereich Organisation und Planung heraus. Ein spezielles Problem aus diesem Bereich ist das Stundenplan-Problem bzw. das timetabling, welches die zeitliche Anordnung einer Menge von Aktivitäten und Ressourcen unter bestimmten Nebenbedingungen umfaßt.
In vielen Bildungseinrichtungen beschäftigt sich in regelmäßigen Abständen eine Anzahl von Lehrkräften in teilweise wochenlanger Arbeit mit der Erstellung von Stundenplänen. Die Ergebnisse dieser Arbeit stellen häufig nur schwer zumutbare Belastungen für die Lehrer dar und das Verhältnis von Arbeitsaufwand und Nutzen kann generell nicht als zufriedenstellend bezeichnet werden.
Diese Arbeit beschäftigt sich mit der Entwicklung einer praxistauglichen Software für die computergestützte Erstellung von Stundenplänen. Sie soll einerseits den Aufwand bei der Suche nach Stundenplänen vermindern und andererseits die Güte der Stundenpläne verbessern. Speziell soll anhand realer Daten untersucht werden, inwieweit die Software ein vollautomatisches Erzeugen von Stundenplänen ermöglicht. Schwerpunkte dieser Arbeit sind die Modellierung der Strukturen von Stundenplänen und die Entwicklung und Anwendung geeigneter Optimierungs-Verfahren für die automatische Stundenplan-Erstellung.

Im Zuge ständig steigender Rechenleistung wurden Ende der sechziger Jahre diesseits und jenseits des Atlantiks zeitgleich Optimierungsverfahren entwickelt, die unter dem Begriff evolutionäre Algorithmen zusammengefaßt werden. Diese Algorithmen wurden in Anlehnung an die Funktionsprinzipien der natürlichen Evolution entworfen. Sie werden mit Erfolg bei Problemen angewandt, die mit klassischen Optimierungsverfahren nicht lösbar sind. Ein solches Problem ist das Stundenplan-Problem.
Eine Charakterisierung der in der Software benutzten Algorithmen als evolutionäre Algorithmen kann aufs Neue die Mächtigkeit und Universalität dieser Verfahren unterstreichen und einen Anreiz darstellen, auch scheinbar wesensfremde Problemklassen nach dem Vorbild der natürlichen Evolution zu bearbeiten.

2. Allgemeine Problemstellung - das Stundenplan-Problem

Im Prinzip kann man nicht von *dem* Stundenplan-Problem (SPP) sprechen. Vielmehr läßt sich eine Reihe von Problemen klassifizieren, die in der Praxis in unterschiedlich strukturierten Institutionen wie etwa Schulen oder Universitäten in den verschiedensten Kombinationen auftreten können. Diese Probleme können in folgende Gruppen unterteilt werden [Bar]:

a) FACULTY TIMETABLING
   Eine Menge von Lehrern wird entsprechend den Fächern, die sie unterrichten können und unter Berücksichtigung ihrer Verfügbarkeiten auf eine Menge von Unterrichtsstunden verteilt.
b) CLASS-TEACHER-TIMETABLING
   Für eine feste Menge von Klassen mit einem unveränderlichen Curriculum und einer festen Menge von Lehrern wird eine Zeitplanung bzgl. der zu unterrichtenden Stunden vorgenommen, derart daß jeder Lehrer und jede Klasse zu jeder Zeit nur einmal Unterricht hat.
c) COURSE SCHEDULING
   Für eine Menge von Schülern, die nicht in Klassen organisiert sind, ist pro Schüler eine Menge von Unterrichtsstunden definiert. Für die Lehrer muß eine Zeitplanung gefunden werden, so daß jeder Schüler zu jeder Zeit nur genau eine Unterrichtsstunde hat.
d) EXAMINATION TIMETABLING
   Eine Menge von Schülern wird einer Menge von Prüfungen zeitlich zugeordnet, so daß kein Schüler gleichzeitig mehr als eine Prüfung hat.
e) CLASSROOM ASSIGNMENT
   Keiner der durch den Unterricht belegten Räume darf gleichzeitig von mehr als einer Unterrichtsstunde belegt sein.

2.1. Komplexität des Stundenplan-Problems

Die praktischen Schwierigkeiten beim Erstellen eines Stundenplans sind in der Komplexität des Problems begründet. In einer Reihe von Untersuchungen wurde die np-Vollständigkeit des SPP gezeigt [Eve],[Coo]. Selbst bis in die Teilprobleme des SPP hinein läßt sich dieser Grad an Komplexität nachweisen [Coo].
Ist es also ein Widerspruch, daß die Entwicklung auf diesem Gebiet fortwährend neue Verfahren hervorbringt, die immer schneller immer bessere Lösungen des SPP liefern?

Die einfachste aber folgenschwerste Erklärung für diese Unstimmigkeit währe: Das SPP ist *nicht* np-vollständig. Die Nachweise der np-Vollständigkeit werden zumeist an stark vereinfachten Modellen vorgenommen, die letztlich nur sehr wenig mit den Problemen gemein haben, denen die Ersteller der Stundenpläne in der Praxis gegenüberstehen. Insbesondere entfällt die Betrachtung der in der Praxis zu beachtenden Nebenbedingungen. Ferner werden die Auswirkungen von Struktur und Umfang der Eingabedaten nicht abgeschätzt. Die eigentliche Frage könnte also lauten: Weisen die realen Modelle aufgrund der Fülle an Nebenbedingungen einen niedrigeren Grad an Komplexität auf? Nebenbedingungen beschränken zwar einerseits den Suchraum, führen aber andererseits nicht automatisch zu einer Steigerung der Effizienz der Suchverfahren. Vielmehr sagen Erfahrungen aus der Praxis, daß Restriktionen das Auffinden von Lösungen gerade *erschweren*. Es ist klar, daß eine zusätzliche Restriktion zu einer Verkleinerung des Suchraumes führt. Entscheidend ist jedoch die Frage, in welchem Maße sich im Vergleich zur reduzierten Größe des Suchraumes die Lösungsmenge verkleinert. Speziell tritt häufig der

Fall ein, daß verschiedene Nebenbedingungen miteinander in Konkurrenz treten - ist die eine Nebenbedingung erfüllt, wird das andere Optimierungs-Ziel nicht erreicht und umgekehrt. Diese Tatsache spricht dafür, daß Nebenbedingungen die Komplexität des SPP sogar *erhöhen*[1].

Der Begriff „Lösung eines SPP" wandelt sich vom theoretischen Informatiker zum Software-Entwickler bis hin zum Stundenplan-Ersteller. Die Abschätzung der Güte eines Stundenplanes wird in der Praxis letztlich von einem subjektiven Standpunkt aus vorgenommen. Der unscharfe Lösungs-Begriff erzeugt eine hohe Zahl zusätzlicher oder „potentieller" Lösungen, so daß die Wahrscheinlichkeit, eine solche Lösung zu finden, eine nur schwer abzuschätzende Erhöhung erfährt[2]. Neben dieser diffizilen Abstufung zwischen Theorie und Praxis läßt sich eine allgemeine „Unschärfe" des Lösungs-Begriffes feststellen. Diese Unschärfe wird durch die Definition sogenannter „weicher" Restriktionen bzw. Nebenbedingungen erzeugt, auf welche im Abschnitt 3.1.3 näher eingegangen wird.

2.2. Bekannte Lösungsansätze der computergestützten SP-Erstellung

Forschungen auf dem Gebiet des timetabling haben in der jüngeren Vergangenheit eine Reihe von Ansätzen und Verfahren hervorgebracht. Der folgende Abschnitt präsentiert eine kurze Übersicht dieser Methoden[Bar].

Exakte Methoden

Der Versuch, Methoden zur Bestimmung *optimaler* Lösungen des SPP zu entwickeln, erscheint in Hinblick auf die np-Vollständigkeit des Problems als hoffnungsloses Unterfangen. Und so ist es auch nicht verwunderlich, daß man mit der Anwendung exakter Methoden bisher nur bei stark vereinfachten Teilproblemen des SPP Erfolge erzielen konnte. Mit exakten Methoden könnten Fortschritte erzielt werden, wenn es gelänge, das SPP in unabhängig voneinander optimierbare Teilprobleme zu zerlegen. Die Schwierigkeiten in der Realität resultieren jedoch gerade aus den starken Verflechtungen und Abhängigkeiten der einzelnen Teilprobleme und Nebenbedingungen.
Aus diesen Gründen basieren die gängigen Programme zur SP-Erstellung hauptsächlich auf heuristischen Verfahren.

Heuristiken [Bar S. 32]

Der Einsatz heuristischer Verfahren, sprich die Nutzung von *Arbeitshypothesen*, deutet auf einen nicht unerheblichen Anteil empirischer Elemente in den Algorithmen hin. Mit anderen Worten: Heuristische Verfahren basieren auf den jeweiligen Erfordernissen des speziell bearbeiteten Problems und auf individuellen Entscheidungen. Dieser Umstand bzw. die daraus resultierende Vielfalt der Verfahren erschwert eine sinnvolle Einordnung in verschiedene Kategorien. Bardadym schlägt eine Charakterisierung der Heuristiken nach zwei Gesichtspunkten vor:

[1] In [Eve] wird gezeigt, daß das Warenfluß-Problem (multicommodity integral flow problem) gerade durch das *Hinzufügen* einer Restriktion np-vollständig wird.
[2] Zitat einer Lehrerin: „Man kann sich einen Plan auch schön gucken."

1. „Simulation of Handmade Timetables' Construction"

Die Idee ist so simpel wie naheliegend, denn der Sinn einer Stundenplan-Software ist es vornehmlich, den Arbeitsaufwand zu vermindern. Die Simulation der manuellen Stundenplan-Erstellung greift auf die Vielzahl der Techniken aus der Praxis zurück. Ein Computer kopiert dann zwar lediglich die Aktivitäten, die ein Lehrer an der Stundentafel ausführt, benötigt dafür jedoch nur einen Bruchteil der Zeit und genau das ist eben der Sinn der Sache. Es ist zweifellos vorteilhaft, auf die Erfahrungen der mit der Plan-Erstellung beauftragten Lehrkräfte zurückzugreifen. Allerdings birgt eine Beschränkung auf die Simulation manueller Techniken die Gefahr in sich, nicht alle Möglichkeiten des Computers auszunutzen.

2. „Decision Rules and Preference Rules"

Die Optimierung eines Planes erfolgt über dessen Veränderung in kleinen Schritten. Eine der einfachsten Regeln, um zu entscheiden, welcher Optimierungs-Schritt vorzunehmen ist, ist eine *zufällig* ausgewählte Verschiebung in der Plan-Struktur vorzunehmen. Häufiger ist eine Aufstellung von heuristischen Entscheidungs-Regeln oder Ziel-Funktionen zu finden, die die Steuerung der Optimierungs-Schritte bestimmt.

Metaheuristiken [Bar S. 33]

Ein Hauptproblem bei Optimierungs-Problemen, insbesondere bei denen, die mit klassischen Verfahren nicht lösbar sind, ist die unerwünschte Konvergenz gegen lokale Extrema. Auch beim SPP treten diese Schwierigkeiten auf, denn die Anzahl gültiger Lösungen und damit auch die Anzahl lokaler Extrema ist i. Allg. sehr hoch.
Metaheuristische Verfahren, zeichnen sich durch die Fähigkeit aus, lokale Extrema während des Optimierungs-Prozesses verlassen zu können. Dabei werden die oben beschriebenen heuristischen Verfahren mit leistungsfähigen Strategien kombiniert, die die Optimierungs-Forschung in den letzten drei Jahrzehnten hervorgebracht hat (siehe auch 4.4.1, 4.4.2 und 5.5).

Interaktives timetabling [Bar S. 34]

Es gibt keine perfekte Software zur automatischen Erstellung von Stundenplänen. Heuristiken sind zwar der manuellen Plan-Erstellung entlehnt, die Simulation des gesunden Menschenverstandes liegt jedoch noch in weiter Ferne. Oftmals liefern Intuition und die Fähigkeit des Menschen, die Plan-Strukturen als Ganzes überblicken zu können Lösungswege, die sich dem Computer nicht erschließen. Aus diesem Grund stellt interaktives timetabling, also die Einbeziehung des Menschen in die computergestützte SP-Erstellung, eine potentielle Effektivitäts-Steigerung bei der Lösungsfindung dar.

Integration von interaktivem timetabling und Optimierungs-Algorithmen [Bar S. 35]

Eine Zusammenführung von Heuristiken, Metaheuristiken und interaktiven Techniken bietet die besten Voraussetzungen für eine effektive computergestützte SP-Erstellung.

3. Planung

Ein einfacher geradliniger Weg für die Entwicklung komplexerer Software ist noch nicht gefunden worden und auch in naher Zukunft nicht in Sicht. So kann denn auch bei dem hier beschriebenen Programm nur schwer zwischen Planungs- und Implementations-Phasen unterscheiden werden. Vielmehr fand während der Entwicklung eine ständige Rückkopplung zwischen Problemdefinition, Erarbeitung des Lösungsmodells, Implementierung und Testphasen statt. Dieser „dynamische Entwicklungsprozeß“ läßt sich einerseits natürlich der traditionellen Motivationslosigkeit der Software-Entwickler gegenüber gewissenhaften Überlegungen in der Planungsphase zuschreiben. Andererseits ist er aber auch eine Folge der Komplexität des SPP - viele Ideen sind während der ersten Optimierungs-Tests entstanden bzw. haben sich als nicht praktikabel erwiesen, schlecht durchdachte Algorithmen erzeugten unerwartete Daten-Konstellationen, die ein Erreichen der Optimierungs-Ziele unmöglich machten u.s.w.. Auch Probleme oder Schwächen der Programmbedienung treten oft erst bei Tests an Prototypen zu tage.
Die Planung orientiert sich an

- Erfahrungs-Berichten von Lehrern
- technischen Voraussetzungen und vorhandene Ressourcen
- einem Prototyp, der im Rahmen eines Seminars entwickelt wurde
- Publikationen aus dem Bereich timetabling

3.1. Problemanalyse und Begriffsdefinitionen

Die Problemanalyse in der Planungsphase ist aufgrund der Vielzahl an Schultypen und damit der breiten Palette an möglichen Anforderungen an eine Software zur computergestützten SP-Erstellung mit Schwierigkeiten verbunden. Darüber hinaus sind die individuellen Wünsche an den Schulen nur schwer vorherzusagen. Durch den eingeschränkten zeitlichen und personellen Rahmen für die Erstellung der Software muß also auch das angestrebte Verhältnis zwischen Aufwand und Nutzen hinterfragt werden.

Als Ausgangspunkt der Problemanalyse werden im folgenden einige Schul-Szenarien vorgestellt, die Repräsentanten von Problemklassen darstellen und/oder bestimmte Nebenbedingungen definieren.

1. Lehrer Leibnitz gibt pro Woche drei Stunden Unterricht in Mathematik bei der Klasse 10b. Keine der Stunden darf am selben Tag stattfinden.
2. Lehrer Goethe gibt pro Woche drei Stunden Unterricht in Deutsch bei der Klasse 10b. Eine dieser Stunden findet Dienstags in der ersten Stunde im Raum 001 statt.
3. Lehrer Einstein unterrichtet die Klasse 10b in Physik. Von den drei Stunden pro Woche, die alle im Physikraum stattfinden müssen, folgen zwei direkt aufeinander.
4. Aufgrund der niedrigen Klassenstärken werden die Klassen 2a und 2b für den Werken-Unterricht bei einem Lehrer zusammengelegt. Dieser Lehrer steht nur Dienstags bis Freitags zur Verfügung.
5. Ab der 7. Klasse müssen alle Schüler eine weitere Fremdsprache wählen. Es wird Einführungs-Unterricht in Spanisch, Französisch und Portugiesisch angeboten. In einer Stunde pro Woche teilen sich die drei 7. Klassen entsprechend der Wahl der einzelnen Schüler auf die drei Sprach-Stunden auf. Die drei Stunden finden alle zur selben Zeit und in den Räumen 001, 002 und 003 statt.

6. In den Sportstunden der Klasse 10a wird eine Aufteilung in Jungen und Mädchen vorgenommen. Je ein Sportlehrer ist für einen Teil verantwortlich. Der Unterricht findet in der Sporthalle statt. Die Klasse 7a hingegen hat ihren Sportunterricht nur bei einem Lehrer in der selben Sporthalle.
7. Wenn es das Fach zuläßt, finden alle Stunden der Klasse 8a in einem bestimmten Raum statt - dem Klassenraum.
8. Für alle Klassen soll realisiert werden, daß die Schule Montags und Freitags möglichst früh zu Ende ist.
9. Die Schulleitung weist jedem Lehrer eine Bereitschaftsstunde zu, so daß stets, wenn möglich, ein Lehrer zur Verfügung steht.
10. Die Musikstunden, die Lehrer Mozart bei der Klasse 6a gibt, dürfen frühestens in der 5. Stunde stattfinden - eine davon unbedingt am Montag.
11. Lehrer und Klassen haben so wenig Freistunden wie möglich.
12. Die Stunden einer jeden Klasse sind möglichst gleichmäßig auf die Unterrichtstage verteilt. Gleiches gilt für Lehrer.
13. Die 8. Klassen sollten möglichst nicht Physik und Chemie an einem Tag haben.
14. Die 8. bis 10. Klassen haben stets bei zwei Sport-Lehrern Unterricht (Jungs und Mädchen getrennt), während alle jüngeren Klassenstufen bei nur einem Lehrer haben. In letzterem Fall wird die Turnhalle geteilt und zwei Klassen haben gleichzeitig Sport.

Die folgende Tabelle enthält eine Zuordnung der Szenarien zu den Problemklassen und Nebenbedingungen:

		Szenarios														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Problem-klassen	**Room Assignment**	×	×	×	×	×	×	×	×	×	×				×	
	Class-Course-Scheduling				×		×								×	
	Class-Teacher-Timetabling	×	×	×		×		×		×						
Neben-bedin-gungen	**Prescheduling**		×			×					×					
	Verfügbarkeiten				×				×							
	Trennungsfehler	×												×		
	Mehrfachstunden			×												
	Klassenräume							×								
	Freistunden											×				
	Stundenverteilung												×			

Tabelle 1: Szenarios für Problemklassen und Nebenbedingungen

Erläuterungen zu Problemklassen und Nebenbedingungen werden in den Abschnitten 3.1.2 und 3.1.3 vorgenommen.

3.1.1. Begriffsbildungen

Voraussetzung für eine korrekte Beschreibung von Problemklassen und Nebenbedingungen und für die Modellierung und Implementierung der Software ist eine klare Begriffsbildung. Die im folgenden definierten Begriffe und deren formale Entsprechungen im Abschnitt 3.2 sind letztlich die Vorlage für eine Reihe von C++ - Klassen in der Implementierung.

STUNDEN
Die Stunde ist die kleinste Organisationsstruktur.
Allgemein gilt: An einer Stunde sind genau ein Lehrer und mindestens eine Klasse beteiligt. Die Beschränkung auf einen Lehrer impliziert die Beschränkung auf ein bestimmtes Fach, welches der beteiligte Lehrer unterrichtet. Jede Kombination aus einem Lehrer und damit auch genau einem Fach und einer Anzahl von Klassen definiert einen STUNDEN-TYP. Stunden kommen in einfacher oder mehrfacher Ausführung vor. Die gebräuchlichste Form der MEHRFACHSTUNDE ist die Doppelstunde. Die Größe der Mehrfachstunden ist nur durch die mögliche Stundenzahl einer Klasse bzw. eines Lehrers pro Tag begrenzt.

EINHEITEN
Eine Einheit ist eine zeitlich zusammenhängende Menge von Stunden gleichen Typs. Eine Mehrfachstunde ist damit automatisch ***eine*** Einheit. Einheiten sind die der Stunde übergeordnete Organisationsstruktur. Jeder Einheit läßt sich ein Raum und/oder eine Zeit zuweisen.

FACHSTUNDEN
Die charakteristische Eigenschaft einer Fachstunde ist die Zuordnung von genau einem Lehrer zu genau einer Klasse in einem bestimmten Fach. Eine Fachstunde ist also das, was man unter einer „klassischen" Unterrichtsstunde verstehen könnte.

KURSE UND KURSSTUNDEN
Eine Kursstunde ist Bestandteil eines Kurses. Charakteristisch für ein Kurs ist, daß eine beliebige Anzahl von Lehrern einer ebenso beliebigen Zahl von Klassen zugeordnet werden kann. Dabei entspricht die Anzahl der involvierten Fächer der Zahl der Lehrer. Eine Fachstunde kann demzufolge als spezielle Kursstunde aufgefaßt werden, wenn nur genau ein Lehrer und genau eine Klasse beteiligt sind[3].
Entsprechend der Anzahl der Lehrer (und damit der Fächer) läßt sich ein Kurs in mehrere Stunden aufspalten, die jeweils einem bestimmten Stunden-Typ angehören. Die Strukturierung der Stunden eines bestimmten Typs in Einheiten ist für alle Lehrer gleich, da alle Stunden einer Einheit zeitlich parallel stattfinden. Diese Stunden werden daher auch als zeitparallele Kursstunden bezeichnet.

3.1.2. Problem-Klassen

Classroom Assignment ... bezeichnet die Anbindung einer jeden Unterrichtsstunde an einen Raum und ist deshalb in den meisten Szenarios wiederzufinden. Der Frage, welche Stunden welchen Räumen zugeordnet werden können, widmet sich u.a. der Abschnitt 3.2.

Class-Teacher-Timetabling ... umschreibt die Forderung, für jede Stunde eine Konstellation Lehrer-Klasse-Fach zu finden.

Class-Course-Scheduling ... ist eine in dieser Arbeit entwickelte Form der vom Class-Teacher-Timetabling abweichenden Zuordnung von (mehreren) Lehrern zu (mehreren) Klassen.

[3] Diese Beziehung wird in der formalen Problemstellung im folgenden Abschnitt noch konsequenter definiert.

3.1.3. Nebenbedingungen

Nebenbedingungen, auch als Restriktionen bezeichnet, werden entsprechend den Gegebenheiten an den Schulen in zwei Kategorien unterteilt - in „harte“ und „weiche“ Restriktionen. Während harte Nebenbedingungen ausnahmslos erfüllt sein müssen, gilt es bei den weichen Restriktionen eine möglichst gute Annäherung an ein optimales (wie auch immer definiertes) Ergebnis zu erzielen. Einige Beispiele aus der Praxis der Stundenplan-Erstellung sind in der folgenden Tabelle aufgelistet:

Nebenbedingung	Szenario	Restriktions-Typ	
		hart	weich
Verfügbarkeiten	Lehrer Meier hat Montags in der ersten und zweiten Stunde einen Weiterbildungskurs und steht in dieser Zeit daher nicht zur Verfügung.	×	
Freistunden	Klassen haben keine Freistunden.	×	
Freistunden	Lehrer haben so wenig Freistunden wie möglich.		×
Stundenverteilung	Die Stunden einer jeden Klasse sind möglichst gleichmäßig auf die Unterrichtstage verteilt.		×

Tabelle 2: Beispiele für Restriktions-Typen

Prescheduling: ... umfaßt die explizite Festlegung für die zeitliche und/oder räumliche Plazierung von Unterricht.

Verfügbarkeiten: Lehrer, Klassen und Räume sind verfügbar, wenn sie zu einer gegebenen Zeit noch nicht in den Stundenplan eingebunden sind und wenn sie zu dieser Zeit nicht von der Planung ausgeschlossen wurden.

Trennungsfehler: ... bezeichnet das Ziel, Fach-gleiche Stunden einer Klasse auf verschiedene Tage der Woche zu verteilen oder aber bestimmte Fächer-Kombinationen für eine Klasse an einem Tag zu vermeiden.

Mehrfachstunden: ... sind oft nur schwer in einen Stundenplan zu integrieren und stellen daher eine separate Nebenbedingung dar.

Klassenräume: Möglichst viele Stunden einer Klasse finden in genau einem Raum statt - dem Klassenraum.

Freistunden: ... ‚auch als LEERSTUNDEN oder Springstunden bezeichnet, sind (zumeist unerwünschte) zeitlich zusammenhängende unbelegte Unterrichts-Zeiten eines Tages, die von belegten Stunden eingeschlossen sind[4].

Stundenverteilung: Sowohl für Klassen, als auch für Lehrer, wird eine möglichst homogene Verteilung der gesamten Stunden pro Woche auf die einzelnen Unterrichts-Tage angestrebt.

[4] Die Beschreibung dieser Nebenbedingung ist eine Ergänzung der Begriffs-Definitionen in Abschnitt 3.1.1.

3.1.4. Zusammenfassung

Zusammenfassend läßt sich feststellen:

a) Das Gesamtproblem ist eine Kombination aus **Class-Teacher-Timetabling** und (Class)**Room Assignment**. Außerdem ist eine spezielle Variante des **Course Scheduling** inbegriffen, welches in dieser Arbeit mit **Class-Course-Scheduling** bezeichnet wird. Weiterhin ist die Berücksichtigung von zeitlichen **Verfügbarkeiten** für Lehrer, Klassen und Räume, **prescheduling** von Unterrichtsstunden und eine Anzahl weiterer **Nebenbedingungen** inbegriffen.
   Die Unterteilung in Problem-Klassen und Restriktionen dient eher als Mittel, um das Gesamt-Problem übersichtlicher darzustellen und stellt keine Rangfolge entsprechend Bedeutung oder Schwierigkeitsgrad dar. Nebenbedingungen können zwar als Ergänzung der Problem-Klassen verstanden werden. Letztlich sind es jedoch die Restriktionen, insbesondere die harten Typs, deren Erfüllung über die Verwendbarkeit eines Planes entscheidet.
b) Nicht unterstützt werden:
   - Kurs-Systeme, wie sie zumeist an Gymnasien zu finden sind
   - die Minimierung der zurückzulegenden Wege von Schülern und Lehrern zwischen den Unterrichtsstunden, falls die Schule auf mehrere Gebäude verteilt ist
   - Mehrfachstunden, deren einzelne Teile aus Unterricht unterschiedlichen Stunden-Typs bestehen
   - Planungs-Formen, die nicht auf einem 1-Wochen-Rhythmus basieren

Die fehlende Unterstützung der in Punkt b) aufgeführten Unterrichts-Formen bzw. Schultypen ist ein Tribut an die begrenzten zeitlichen und personellen Ressourcen. In eventuell folgende Programmversionen sollen jedoch einige dieser Varianten integriert werden.

3.2. Formale Problemstellung

Es erfolgt eine formale Darstellung des Problems. Die Darstellung ergibt sich aus den im vorherigen Abschnitt vorgenommenen Begriffs-Definitionen und den beschriebenen realen Anforderungen an das Programm. Mehrfachstunden und damit auch Einheiten werden nicht betrachtet, um ein unnötiges Aufblähen der formalen Problemstellung zu vermeiden. Ziel dieses Abschnittes ist es vornehmlich, den Charakter der Problemklassen Classroom Assignment, Class-Course-Scheduling und Class-Teacher-Timetabling näher zu beleuchten und eine exaktere Beschreibung der in die Problemstellung einbezogen Begriffe LEHRER, KLASSEN, RÄUME, FÄCHER, STUNDEN, ZEITEN und VERFÜGBARKEITEN vorzunehmen.

Mengen werden im folgenden mit einem großen (Anfangs-)Buchstaben gekennzeichnet. Variablen mit kleinen (Anfangs-)Buchstaben sind Elemente von Mengen.
Neue Begriffe werden in Kaptitälchen-Schreibweise dargestellt.

Folgende Mengen werden definiert:
$L = \{l_0, ..., l_{LA}\}$ - Menge aller Lehrer
$K = \{k_0, ..., k_{KA}\}$ - Menge aller Klassen
$R = \{r_0, ..., r_{RA}\}$ - Menge aller Räume
$F = \{f_0, ..., f_{FA}\}$ - Menge aller Fächer

$S = \{s_0, ..., s_{SA}\}$ - Menge aller Stunden pro Woche, $S^* = S \cup s_\phi$, wobei s_ϕ eine Leerstunde ist
$Z = \{0, ..., 90\}$ - Menge aller Zeiten; $Z^* = Z \cup z_\phi$, wobei z_ϕ eine undefinierte Zeit bedeutet

Für jeden Lehrer, jede Klasse und jeden Raum existiert eine Menge von Zeiten, die die NICHT-VERFÜGBARKEIT beschreibt:
$NVL_l \subseteq Z$ - Menge von Zeiten, in denen Lehrer l nicht verfügbar ist
$NVK_k \subseteq Z$ - Menge von Zeiten, in denen Klasse k nicht verfügbar ist
$NVR_r \subseteq Z$ - Menge von Zeiten, in denen Raum r nicht verfügbar ist

Zwei weitere wichtige Variablen, die einen Zeitrahmen definieren: Brauchen wird das?
$Tage = 7$ Unterrichtstage pro Woche
$Stunden = 13$ maximale Stundenzahl pro Tag für alle Klassen, Lehrer und Räume
Es gilt: $|Z| = Tage \cdot Stunden = 91$.

Jedem Lehrer ist eine Menge von Fächern zugeordnet, die er unterrichten kann:
$F_l = fae(l) \subseteq F, \; l \in L$.

Jedem Fach ist eine Menge von Räumen zugeordnet, in denen das Fach unterrichtet werden kann:
$R_f = rau(f) \subseteq R, \; f \in F$.

Eine KURS-DEFINITION ist ein Tupel (l, f, RT) mit $l \in L$, $f \in fae(l)$, $RT \subseteq rau(f)$. Kurs-Definitionen werden also durch die Lehrer und deren Fächer bestimmt. Ferner impliziert die Teilmengen-Beziehung zwischen RT und $rau(f)$ die Möglichkeit, einen einzigen speziellen Raum zu definieren und dadurch die spätere Unterrichtsstunde auf diesen Raum festzulegen (prescheduling; siehe auch Begriff „raumfest“). ?
Die Menge aller Kurs-Definitionen ist $AKD = \{(l, f, RT) \mid l \in L, f \in fae(l), RT \subseteq rau(f)\}$.

Eine STUNDEN-DEFINTION ist eine Menge von Tupeln $(Klassen, kd, zeit)$ mit $Klassen \subseteq K$, $kd \in AKD$, $zeit \in Z^*$, wobei für alle Tupel der Stunden-Definition $Klassen$ und $zeit$ übereinstimmen. In Hinblick auf die Praxis, entspricht je ein Tupel einer Stunden-Definition mit $|Klassen| = 1$ der Beschreibung einer normalen Unterrichtsstunde, denn mit der enthaltenen Kurs-Definition setzt sich das Tupel aus den notwendigen Informationen Klasse, Lehrer, Fach, Raum (bzw. Räume) und Zeit zusammen. ?
Für $zeit \neq z_\phi$ gilt die Unterrichtsstunde als ZEITFEST. Für $kd = (l, f, RT)$ und $|RT| = 1$ wird die Unterrichtsstunde als RAUMFEST bezeichnet.

Stunden-Definitionen lassen sich entsprechend ihrer Tupel - Anzahl auf STUNDEN - Tupel der Form $(Klassen, l, f)$ reduzieren. ?
Jede (Unterrichts-)Stunde geht im Rahmen dieses Modells also aus einer Kurs-Definition hervor. Auch in der Praxis läßt sich eine normale Stunde als spezielle Kursstunde auffassen.

Die Menge aller Stunden sei S. $S^* = S \cup s_\phi$, wobei s_ϕ die leere Stunde symbolisiert.

Ein RAUMPLAN ist eine Menge von 91 Tupeln (z, s), mit $z \in Z$ und $s \in S^*$. Jedes z aus Z ist in genau einem Tupel pro Raumplan enthalten. Jedes dieser 91 Tupel fungiert als Container

für die Aufnahme einer Stunde. Für ein Tupel (z,s) mit $s = s_\phi$ gilt der entsprechende Raum als unbelegt.

Die Menge aller Raumpläne sei ARP.

Mit dem Begriff Raumplan läßt sich nun ein Stundenplan (SP) definieren:
Ein STUNDEPLAN SP ist die Menge aller Tupel (r, RP_r) mit $r \in R$ und $RP_r \in ARP$.

Als Beispiel mögen folgende Mengen gegeben sein:
$K = \{7a, 7b, 7c\}$, $L = \{Müller, Meier, Schmitt, Bonaparte\}$, $R = \{BioR, PhR, ChR, R4, R5\}$
$F = \{Ph, Bio, Ch, E, Fr, D\}$

und es möge gelten:
$rau(Bio) = \{BioR\}$, $rau(Ph) = \{PhR\}$, $rau(Ch) = \{ChR\}$
$rau(E) = rau(Fr) = rau(D) = \{R4, R5, R6\}$
$fae(Müller) = \{Bio, Ch\}$, $fae(Meier) = \{Bio, Ph\}$,
$fae(Schmitt) = \{Ch, D, E\}$, $fae(Bonaparte) = \{E, Fr\}$.

Für alle Lehrer, Klassen und Räume sei die Menge der nicht verfügbaren Stunden $NV = \phi$.

Die Menge der Stunden-Definitionen $S = \{s_1, s_2, s_3, s_4\}$ setze sich wie folgt zusammen:

$s_1 = \{(\{7a\}, (Schmitt, D, rau(D)), z_\phi)\}$ besteht nur aus einem Tupel und bedeutet, daß Lehrer Schmitt bei Klasse 7a das Fach D in einem der für das Fach D vorgesehenen Räume unterrichtet und die Zeit für diese Stunde noch nicht feststeht. Diese einfachste Ausprägung einer Stunde wird im folgenden auch als FACHSTUNDE bezeichnet. Fachstunden sind Stunden der Form: ein Lehrer/ ein Fach/ eine Klasse.

$s_2 = \{(\{7b\}, (Schmitt, D, \{R5\}), 3)\}$ bedeutet, daß Raum R5 und Zeit 3 für diese Fachstunde definiert wurden. s_2 ist also zeitfest und raumfest.

$s_3 = \left\{\begin{array}{l}(\{7c\}, Schmitt, E, rau(E), 5), \\ (\{7c\}, Bonaparte, Fr, rau(Fr), 5)\end{array}\right\}$ bedeutet, daß für die Klasse 7*a* zeitgleich Unterricht im Fach *E*, bei Lehrer *Schmitt* und im Fach *Fr* bei Lehrer *Bonaparte* durchgeführt wird, wobei die Zeit für die Stunde bereits feststeht. Die Klasse 7*a* wird also zur gegebenen Zeit geteilt - die eine Hälfte hat Französisch bei Bonaparte, die andere Englisch bei Schmitt. Dieser Typ einer Stunden-Definition beschreibt Unterricht, der in Form von KURSSTUNDEN stattfindet. Kursstunden sind Stunden der Form: x Lehrer/ x Fächer/ y Klassen.

$s_4 = \left\{\begin{array}{l}(\{7a, 7b, 7c\}, (Schmitt, Ch, rau(Ch)), z_\phi), \\ (\{7a, 7b, 7c\}, (Meier, Ph, rau(Ph)), z_\phi), \\ (\{7a, 7b, 7c\}, (Müller, Bio, rau(Bio)), z_\phi)\end{array}\right\}$ ist ein Beispiel für eine klassische Kursstunde.
Die Schüler der drei Klassen können für eine Stunde zwischen drei Fächern wählen. Jede Klasse teilt sich also in drei Teile und wird den jeweiligen Fächern bzw. Lehrern zugewiesen.

Die Zuordnung der Stunden-Tupel einer Stunden-Definitionen in die Raumpläne erfolgt durch Wahl einer Zeit und eines Raumes aus der Menge der verfügbaren Räume. Jedes Stunden-Tupel „kennt“ die Menge der in Frage kommenden Räume und muß deshalb nicht aus der gesamten Raum-Menge wählen. Dieser Umstand ist von großer Bedeutung für die spätere Implementierung.
Eine Funktion $SetzTest : K \times L \times F \times R \times Z \rightarrow \{0,1\}$ liefert die Entscheidung, ob das Plazieren der Stunde an die gewählte Position die GÜLTIGKEIT des Planes erhält oder nicht. Ein Plan ist gültig, wenn folgende Bedingungen gelten:

a) Jeder Stunde ist genau eine Zeit zugeordnet. Das heißt, alle Stunden werden genau einmal verplant.
b) Es liegen keine Überschneidungen bei Räumen und Lehrer vor. Zu jeder Zeit darf jeder Raum und jeder Lehrer stets nur einmal vergeben werden.
c) Zeitliche Überschneidungen für Klassen sind im Rahmen einer Stunden-Definition erlaubt bzw. zwingend. Entsprechend der Anzahl der Tupel (= Anzahl der Lehrer) einer Stunden-Definition werden jeder beteiligten Klasse mehrere zeitgleiche Tupel in den Raumplänen zugeordnet.
d) Die Verfügbarkeiten müssen berücksichtigt werden.
e) Vordefinierte Zeiten und Räume müssen berücksichtigt werden.

Ausgehend von den Beispiel-Daten ist *ein* gültiger Stundenplan die Menge der Raumpläne

$$SP = \{RP_{ChR}, RP_{BioR}, RP_{PhR}, RP_{R4}.RP_{R5}\}$$

$$= \left\{ \begin{array}{l} (ChR, \{(0, s_\phi), (1, \{7a,7b,7c\}, Schmitt, Ch), (2, s_\phi), (3, s_\phi), ...\}), \\ (BioR, \{(0, s_\phi), (1, \{7a,7b,7c\}, Müller, Bio), (2, s_\phi), (3, s_\phi), ...\}), \\ (PhR, \{(0, s_\phi), (1, \{7a,7b,7c\}, Meier, Ph), (2, s_\phi), (3, s_\phi), ...\}), \\ (R4, \{(0, s_\phi), (1, s_\phi), (2, \{7a\}, Schmitt, D), (3, s_\phi), (4, s_\phi), (5, \{7c\}, Schmitt, E), (6, s_\phi), (7, s_\phi), ...\}), \\ (R5, \{(0, s_\phi), (1, s_\phi), (2, s_\phi), (3, \{7b\}, Schmitt, D), (4, s_\phi), (5, \{7c\}, Bonaparte, Fr), (6, s_\phi), (7, s_\phi), ...\}) \end{array} \right\}$$

Wie am Anfang dieses Abschnittes bereits angedeutet, beschreibt diese formale Definition das in die Software umgesetzte Modell nicht vollständig. Im Interesse einer höheren Flexibilität des Programmes wurden in der Implementierung eine Reihe von Definitionen erweitert[5], andere wurden aus Gründen der Effizienz bei der Optimierung eingeschränkt. Teilweise verändert der Programm-Nutzer selbst durch seine Entscheidungen die Definitionen[6].

[5] Es z.B. möglich, für jede Unterrichtsstunde einen **beliebigen** Raum vorzudefinieren.
[6] Die Platzierung von Kursstunden an das Ende eines Tages, welche die Optimierung negativ beeinflußt, kann z.B. unterbunden werden.

3.3. Leistungsfähigkeit und Funktionsumfang

Die Planung von Leistungsfähigkeit und Funktionsumfang der Software orientiert sich an den in der Praxis zu beobachtenden Arbeitsschritten bei der Stundenplan-Erstellung. Die folgende Grafik beschreibt die Plan-Erstellung als einen Arbeits-Prozeß, welcher in drei bzw. vier Phasen abläuft.

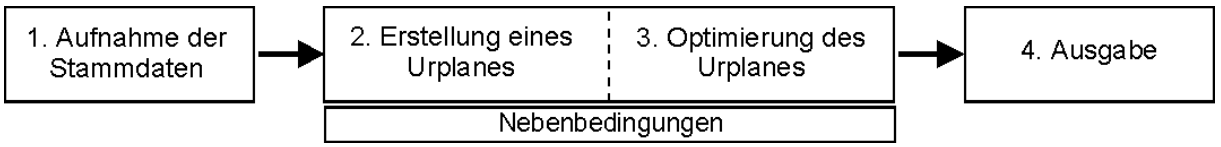


Abbildung 1: Planungs-Phasen in der Praxis

Phase 1 erfordert in der Praxis der manuellen Stundenplan-Erstellung nur einen relativ geringen Arbeitsaufwand, da die Zuordnung von Lehrern und Fächern zu den Klassen bereits im Vorfeld erarbeitet wird. Allerdings müssen i. Allg. umfangreiche Listen oder Tabellen erstellt werden, anhand derer u.a. die Vollständigkeit der Planungsdaten überwacht wird.

Die Phasen 2 und 3 nehmen den größten Teil der Zeit in Anspruch. Phase 2 beinhaltet die Erstellung eines ersten gültigen Planes (Urplan-Erstellung). Die Schwierigkeiten bei der Urplan-Erstellung liegen vor allem in der Vermeidung von zeitlichen Überschneidungen. In Phase 3 wird die Güte dieses Planes schrittweise verbessert. Die Arbeitsschritte 2 und 3 werden zusätzlich durch die Notwendigkeit erschwert, permanent die Einhaltung einer Fülle von Nebenbedingungen berücksichtigen zu müssen.
Die Phasen 2 und 3 werden, wie es die obige Grafik andeutet, wohl nur in den seltensten Fällen strikt nacheinander ausgeführt[7]. Vielmehr gehen diese beiden Phasen in einem Maße zeitlich ineinander über, welches von den individuellen Techniken der Stundenplan-Ersteller bestimmt wird.

Phase 4 umfaßt das Anfertigen der Stundenpläne für alle Lehrer, Klassen und Räume in Papier-Form. Sie erfordert also keine großen schöpferischen Leistungen, nimmt aber trotzdem einen oder zwei Tage in Anspruch, falls keine Hilfsmittel zur Verfügung stehen.

3.3.1. Stammdaten-Verwaltung

Zu den Stammdaten gehören Fächer, Lehrer, Klassen, Räume, Kurse, Unterrichts-Definitionen und einige allgemeine Angaben wie z.B. die maximale Stundenzahl pro Tag oder die Möglichkeit, nullte Stunden zu integrieren.
Es ist kein Einzelfall, daß Schulen in Software investieren, die erst mehrere(!) Jahre später oder überhaupt nicht benutzt werden kann, weil die Bedienung einfach zu kompliziert ist. Erfahrungsberichte von Lehrern, die bereits mit anderen Stundenplan-Programmen gearbeitet

[7] Oftmals haben Lehrer mit der Schwierigkeit zu kämpfen, daß sich noch während der Planerstellung viele Nebenbedingungen ändern. Daher ist die Einteilung in zwei Phasen für die Praxis eher eine Idealisierung. Z.B. erhalten manche Lehrer erst sehr spät die Termine für ihre wöchentlichen Weiterbildungs-Kurse oder die Planung muß wegen gemeinsamer Nutzung der Sporthalle mehrmals mit dem Stundenplan einer anderen Schule abgestimmt werden.

haben, offenbaren immer wieder dieses Problem. Ein wichtiges Ziel bei der Stammdaten-Verwaltung sollte also eine möglichst einfach benutzbare graphische Oberfläche sein. Um diese Zielsetzung zu erreichen, werden einige grundlegenden Eigenschaften und Funktionalitäten formuliert:

- Die einzelnen Bestandteile der Stammdaten, also Lehrer, Klassen u.s.w., erhalten jeweils *ein* separates Formular bzw. Fenster.
  Diese Trennung unterstützt den Versuch, die Formulare möglichst klein und übersichtlich zu halten.
- Die Navigation innerhalb der Stammdaten Formulare und deren Benutzung ist, soweit möglich, standardisiert in Layout und Funktionalität.
  Jedes Formular enthält z.B. die Buttons **Neu**, **Umbenennen**, **Löschen** und **Schließen**, die sich dann auf den jeweiligen Stammdaten-Typ beziehen. Im Lehrer-Formular ruft der Button **Neu** also einen Dialog zur Erzeugung eines neuen Lehrers hervor. Die Navigation (das „Blättern“) durch die Menge der Lehrer ist identisch mit der Navigation im Klassen-Formular u.s.w..
- Alle Formulare sind frei auf dem Bildschirm plazierbar.
  Dadurch hat der Nutzer die Möglichkeit, zwei oder mehrere Formulare nebeneinander auf dem Bildschirm zu betrachten. Allerdings setzt dies bei bestimmten Formularen eine akzeptable Bildschirm-Auflösung voraus[8].
- Alle Formulare lassen sich über eine Speedbar aufrufen bzw. in Vordergrund bringen.
- Die Anordnung der Formulare auf dem Bildschirm läßt sich in einer Konfigurationsdatei speichern.
- Soweit möglich, sollten die graphischen Elemente zur Dateneingabe auf Standard-Windows-Komponenten beschränkt sein.
- Für jedes Formular steht ein Kontext-sensitives Hilfe-System zur Verfügung.

3.3.2. Stundenplan-Erstellung

Erfahrungsberichte von Lehrern besagen, daß es kein perfektes Programm zur automatischen Erstellung von Stundenplänen gibt und es kann kaum davon ausgegangen werden, daß XXX[9] dieses Manko beseitigen wird. Im Normalfall werden existierende Programme von den Schulen als mehr oder weniger praktikable *Unterstützung* der Planerstellung angewandt. An dieser Stelle gilt es also zu analysieren, worin genau eine solche Unterstützung bestehen kann. Die Erstellung von Stundenplänen in der Praxis läßt sich, wie bereits in Abbildung 1 dargestellt, auf zwei grundlegende Phasen reduzieren. Zuerst wird ein zwar gültiger aber „schlechter“ Plan erstellt (der Urplan) und anschließend eine Optimierung dieses Planes vorgenommen.
Für eine automatische Urplan-Erstellung scheint der Computer regelrecht prädestiniert zu sein. Vor allem die bei der manuellen Plan-Erstellung überaus aufwendige Prüfung von zeitlichen Überschneidungen kann automatisiert werden. Weiterhin erhalten die harten Restriktionen durch die strikten Forderungen, durch die sie definiert sind, einen ähnlichen Charakter wie die Problemklassen. Daher sollten diese Nebenbedingungen, soweit möglich, bereits in der Urplan-Erstellung enthalten sein. Dies setzt wiederum die Notwendigkeit voraus, die entsprechenden Restriktionen bereits in der Stammdaten-Phase erfassen zu

[8] Das gleichzeitige öffnen verschiedener Stammdaten-Fenster erfordert einen teilweise komplexen Abgleich der Daten. Vor allem Lösch-Aktionen können weitreichende Folgen haben. Wird z.B. ein Fach gelöscht, müssen alle Lehrer, die dieses Fach unterrichten, aktualisiert werden. Weiterhin wird jeglicher Unterricht in diesem Fach ungültig u.s.w...
[9] Der Name für die Software steht noch nicht fest. Daher wird im folgenden dieses Pseudonym verwendet.

können. Für die computergestützte Plan-Erstellung ergibt sich nun ein leicht verändertes Schema der Arbeitsschritte:

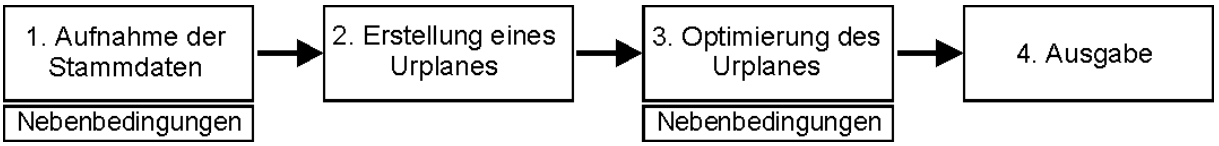


Abbildung 2: Planungs-Phasen in der Software

Die Erstellung des Urplanes ist nun ein separater Arbeitsschritt und kann daher leichter automatisiert werden. Die Schwierigkeit, die sich daraus ergibt, besteht in der Notwendigkeit, auch *nach* der Urplan-Erstellung möglichst viele Restriktionen manipulieren zu können - insbesondere natürlich jene, die bereits in der Stammdaten-Phase festgelegt wurden.
Die Optimierung des Urplanes (Phase 3) beinhaltet die Erfüllung der während der Urplan-Erstellung noch nicht berücksichtigten Nebenbedingungen.

Zusammenfassend lassen sich für die Phasen 2 und 3 folgende anzustrebende Programmeigenschaften formulieren:

- effiziente Algorithmen zur Erstellung und Optimierung von Stundenplänen
- Möglichkeiten zur einfachen manuellen Nachbearbeitung von Plänen[10]

Die Frage, wann und in welchem Umfang der Nutzer in die Erstellung und Optimierung des Stundenplanes als „regelnde Person“ eingreifen kann bzw. muß, konnte in der Planungsphase noch nicht beantwortet werden. Sie betrifft die schwierige Suche nach dem bestmöglichen Kompromiß zwischen Flexibilität und Komfort. Ein hoher Grad an Automatisierung geht auf Kosten der Flexibilität der Software, vereinfacht aber die Benutzung des Programmes, während ein niedriger Grad an Automatisierung mehr Verständnis des Users erfordert, ihm aber gleichzeitig mehr Möglichkeiten eröffnet die Fähigkeiten der Software voll auszunutzen. Wünschenswert sind demzufolge *zwei* Benutzer-Modi, die jeweils den Bedürfnissen nach Flexibilität oder Komfort Rechnung tragen.

3.3.3. Ein - / Ausgabe, Persistenzhaltung

Die Erstellung eines Stundenplanes vollzieht sich in zwei Schritten - der Unterrichtsplanung und der eigentlichen Planerstellung. Die Unterrichtsplanung beinhaltet das Zusammentragen einer Fülle von Daten. Im Einzelnen sind dies Fächer, Klassen, Lehrer, Räume, Kurse und die Menge der Unterrichtsdefinitionen. Diese Daten werden im folgenden als Stammdaten bezeichnet.
Die Stammdaten sind Ursprung und Bestandteil des eigentliche Stundenplanes. Allerdings ist es im Verlauf der Planerstellung oft wünschenswert, bestimmte Teile der Stammdaten nachträglich zu ändern - etwa das Verschieben einer Stunde in einen anderen Raum und deren anschließende Fixierung (raumfest), die zeitliche Fixierung einer Stunde (zeitfest) oder das nachträgliche Sperren eines Lehrers zu einer bestimmten Zeit. Die meisten Änderungen dieser

[10] Im Idealfall ersetzt die Umsetzung der Nachbearbeitungs-Möglichkeit im Programm vollständig die sog. Stundentafel.

Art stellen jedoch nur in Hinblick auf den *aktuell* bearbeiteten Plan eine sinnvolle Nebenbedingung dar. Es stellt sich also die Frage, ob und wie nachträgliche Änderungen der Stammdaten in der Planungs-Phase behandelt werden sollten.
Desweiteren ist es wünschenswert, ausgehend von *einer* Stammdaten-Menge, *verschiedene* Pläne erstellen und persistent halten zu können - etwa um der Lehrerschaft mehrere Pläne zu präsentieren oder um während der Optimierung Zwischenergebnisse festhalten zu können.
Auch das Speichern *mehrerer* verschiedener Stammdaten-Mengen mit unterschiedlich gesetzten Schwerpunkten bei den Nebenbedingungen sollte möglich sein. Alle diese Anforderungen führen zu der Notwendigkeit, zwei verschiedene Dateitypen einzuführen: Stammdaten-Dateien und Plan-Dateien.
An die Plan-Dateien muß insbesondere die Forderung gestellt werden, daß ein persistent gehaltener Plan nach dem Laden weiter bearbeitet werden kann. Es ist erforderlich, auch die Änderungen der Stammdaten während der Planungs-Phase persistent zu halten - kurz gesagt, *der gesamte Programmzustand in einem bestimmten Augenblick der Optimierung sollte reproduzierbar sein*.
Neben dem normalen Speichern in Dateien sind zwei weitere Möglichkeiten der Ausgabe geplant - die direkte Ausgabe an den Drucker und der Daten-Export an Winword. Die individuellen Vorstellungen für das Druck-Layout sind breit gefächert und es scheint unmöglich, diese alle vorherzusehen oder gar in das Programm zu integrieren. Die Ausgabe in Winword soll dem Nutzer die Möglichkeit geben, Layout-Entscheidungen mit möglichst geringen Einschränkungen selbst treffen zu können. Darüber hinaus stellt das Speichern einer Winword-Datei eine weitere Möglichkeit der Persistenzhaltung dar.

3.3.4. Schultypen, Nebenbedingungen, Sonderfälle

Die Frage nach zu unterstützenden Unterrichts-Typen und Nebenbedingungen steht in engem Zusammenhang mit der Problemanalyse und den Begriffsdefinitionen in Punkt 3.1 und entscheidet letztlich über die späteren Einsatzmöglichkeiten der Software.

Schul- bzw. Unterrichtstyp:

Das Programm unterstützt den normalen Unterricht im Klassenverband. Dabei ist es jedoch möglich, aus mehreren beliebigen Klassen Kurse zu bilden. Dies bedeutet, daß den Schülern dieser Klassen entsprechend ihren Interessen ermöglicht wird, sich während einer bestimmten Zeit auf verschiedene Fächer und somit auf entsprechende Lehrer zu verteilen (siehe z.B. Abschnitt 3.1 / Szenario 5).

Mehrfachstunden:

Es ist möglich, Stunden gleichen Faches zeitlich miteinander zu koppeln. Die übliche Form dieser Mehrfachstunden ist die Doppelstunde.

Zeit-Restriktionen:

Betrachtet man einen Stundenplan als Tabelle mit Stunden als Zeilen und Tagen als Spalten, ist für jede Unterrichtsstunde ein rechteckiges Zeitfenster definierbar, in welchem die Stunde stattzufinden hat. Als Spezialfall dieses preschedulings läßt sich jeder Unterricht auf eine ganz bestimmte Zeit exakt vorplanen.
Lehrer, Klassen und Räume können zu beliebigen Zeiten gesperrt werden und stehen dann nicht für die Planerstellung zur Verfügung.

Raum-Restriktionen:

Für jede Unterrichtsstunde kann ein fester Raum vorgegeben werden.
Für jeden Raum kann eine Menge von Fächern angegeben werden, die dort unterrichtet werden dürfen. In Ausnahmefällen kann diese Reglementierung auf Wunsch aufgehoben werden.
Soll eine Klasse während eines Schultages möglichst selten den Raum wechseln müssen, kann dieser Klasse ein sog. Klassenraum zugewiesen werden, in welchem der gesamte Unterricht stattfindet, soweit das entsprechende Fach nicht einen speziellen Raum erfordert. Verteilt sich eine solche Klasse während eines Kurses in verschiedene Räume, kann der Klassenraum auf Wunsch für diese Kursstunden freigegeben werden, falls die Klasse an dem Kurs beteiligt ist.

Fach-Restriktionen:

Es ist zumeist wünschenswert, daß der gesamte Unterricht einer Klasse in einem Fach an verschiedenen Tagen stattfindet (Mehrfachstunden ausgenommen). Für jedes Fach besteht also die Möglichkeit, dies als Planungs-Ziel festzulegen. Weiterhin sind bestimmte Kombinationen von Fächern an einem Tag unerwünscht. Auch hier ist es möglich, die entsprechende Verteilung dieses Unterrichts über die Woche zu planen.

3.3.5. Hilfe, Dokumentation

Wie in Abschnitt 3.3.1 bereits angesprochen, ist die einfache Bedienbarkeit des Programmes von großer Bedeutung. Es kann nicht davon ausgegangen werden, daß die für die Stundenplan-Erstellung verantwortlichen Lehrkräfte an den Schulen erfahren im Umgang mit dem Computer sind. Um dem Nutzer der Software bestmögliche Unterstützung zukommen zu lassen, sollten in jeder Phase der Programm-Benutzung kontextbezogene Hilfetexte zur Verfügung stehen.
Die ersten Schritte bei der Arbeit mit der Software sollten durch ein Tutorial Unterstützung finden.

3.3.6. Extras

Die Software bietet komfortable Funktionen für die Erstellung von Vertretungsplänen.

3.4. Durchführbarkeit

Einführend soll noch einmal auf das Ziel hingewiesen werden, eine Software zu entwickeln, die über den Status eines Prototyps hinausgeht. Ob und wie gut dies gelingt, ist von einigen Faktoren abhängig - von der Frage nach der prinzipiellen Lösbarkeit des Problems, der Zeitplanung und den technischen Voraussetzungen.

3.4.1. Lösbarkeit des Problems

Die Frage nach der Lösbarkeit des Problems läßt sich auf die Frage nach der Effizienz der Optimierungs-Algorithmen reduzieren und die Suche nach diesen Algorithmen ist ein zentraler Gegenstand dieser Arbeit. Prognosen in diese Richtung währen also eher

spekulativer Art. Einige in dieser Arbeit dokumentierte Ideen, Ansätze und Entwurfsentscheidungen basieren jedoch auf einem Programm, welches im Wintersemester 1996/97 im Rahmen des Seminars „Optimierung und Simulation“ an der Universität Potsdam entwickelt wurde [Wei]. Obwohl diese Software nicht über den Status eines Prototyps hinausgeht, setzt sie ein Teilmodell des hier beschriebenen Programmes erfolgreich um. Sie bearbeitet einfache Probleme, die eine Kombination aus Class-Teacher-Timetabling und Classroom Assignment darstellen. Abgesehen von dem Versuch, Freistunden zu minimieren, werden keine Nebenbedingungen berücksichtigt. Insbesondere entfällt die Behandlung von Kursen und Mehrfachstunden vollständig. Erstens dient diese Seminar-Arbeit als Orientierungshilfe bei der Entwicklung der Optimierungs-Algorithmen, deren Grundprinzip des „tabu search“ sich in der aktuellen Arbeit wiederfindet. Zweitens fließen Erfahrungen bei der Urplan-Erstellung in die Software ein (siehe auch 5.2).
Die Erfahrungen bei der Programmierung des Prototyps und die erzielten Ergebnisse rechtfertigen zwar eine optimistische Einstellung in Bezug auf die Lösbarkeit des Problems. Andererseits jedoch sind die Erfolgsaussichten ungewiß, da sich der Prototyp lediglich auf die Problemklassen Class-Teacher-Timetabling und Classroom Assignment beschränkt.

3.4.2. Zeitplanung

Von allen Ressourcen ist es erfahrungsgemäß die Zeit, an der es im Software-Entwicklungsprozeß am meisten mangelt. Von entsprechend großer Bedeutung ist daher die Zeitplanung. Die Lehrer von zwei Schulen erklärten sich im Vorfeld freundlicherweise bereit, die Daten ihrer Unterrichtsplanung zur Verfügung zu stellen. Mit Beginn der 5. oder 6. Woche beginnen die Schulen mit der Planung ihres Unterrichts. Zu diesem Zeitpunkt sollte ein Prototyp existieren, der über eine funktionierende Stammdaten-Verwaltung verfügt und wenn möglich schon in der Lage ist, erste Stundenpläne zu erzeugen. Die Tests an den Schulen dienen weniger der Erstellung von Stundenplänen. Wichtiger erscheint die Aufnahme der Daten für die spätere Entwicklung bzw. Verfeinerung der Optimierungs-Algorithmen. Außerdem könnten in Gesprächen mit den Lehrern Hinweise für die gesamte Gestaltung der Software gesammelt werden.

Wochen	***1** (14.7.)*	***2***	***3***	***4***	***5** (15.8.)*	***6***	***7***
	|→ Eingabe Stammdaten + Persistenzhaltung →						
			|→ Urplanerstellung →				
			Optimierungs-Algorithmen →				
			|→ Hilfesystem →				
			|→ Persistenzhaltung der Plandaten →				
	|→ Nebenbedingungen →						
					|→ Tests an den Schulen →|		
	|→ Hausarbeit →						

Tabelle 3: Zeitplanung

3.4.3. Technische Voraussetzungen

Die wichtigste Voraussetzung für die Programmierung ist die verwendete Entwicklungsumgebung. Für die Entwicklung von XXX wurde der C++ Builder von Inprise (ehemals Borland) verwendet. „Borland C++Builder ist eine objektorientierte visuelle Entwicklungsumgebung für Rapid Application Development (RAD).“[11] und als diese zur Zeit wohl konkurrenzlos unter den C++ Entwicklungsumgebungen. Er ist das C++ - Pendant zu Delphi, verbindet also die Vorteile Komponenten-basierter Programmiertechniken mit der Mächtigkeit der Sprache C++.

Die VCL (Visual Component Library), die Bibliothek des C++ Builder, kapselt die Programmier-Schnittstelle von Windows, vermeidet also weitestgehend die Berührungspunkte mit dem „Alptraum Windows-API“ und ermöglicht so vor allem eine schnelle Entwicklung des äußeren Erscheinungsbildes bzw. des „Look and Feel„ eines Programmes.

Das Hilfe-System wird in Form von HTML-Dokumenten zur Verfügung gestellt. Die Erstellung der Hilfe-Texte erfolgt mit der Web Publishing Software „Net Object Fusion 3.0“.

Für die Bereitstellung der Software an den Schulen und die sichere Aufnahme der anfallenden Daten dient ein Zip-Laufwerk, welches sich problemlos an den Parallelport jedes Windows-Rechners anschließen läßt.

Die Programmierung der Optimierungs-Algorithmen erfordert insbesondere zeitaufwendige Tests. Aus diesem Grund stehen zwei vernetzte Rechner zur Verfügung, um die Programmierung und die Testphasen parallel durchführen zu können.

[11] C++Builder - Dokumentation

4. Darstellung von Lösungsmodell und Lösungsidee

4.1. Programm-Ablauf

Die Überlegungen in Abschnitt 3.3.3 erbrachten die Schlußfolgerung, zwei verschiedene Programm-Zustände und damit auch zwei verschiedene Datei-Typen einführen zu müssen. Diese erzwingt einen bestimmten Programm-Ablauf, welcher wiederum die verschiedenen Datenstrukturen des Programms beeinflußt, die das Thema des nächsten Abschnittes sind. Das folgende Zustands-Diagramm enthält eine grobe Darstellung des Programm-Ablaufs im UML - Stil [Dum].

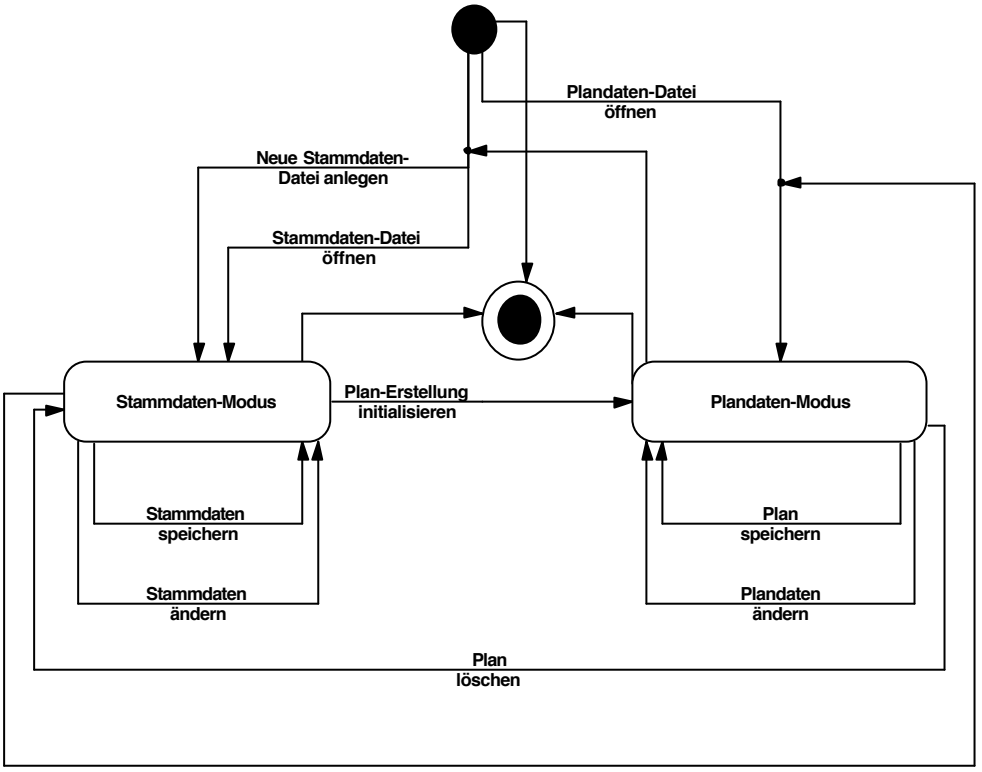


Abbildung 3: Programm-Ablauf und zentrale Programm-Zustände

Hervorzuheben sind die zentralen Programm-Zustände **Stammdaten-Modus** und **Plandaten-Modus** (kurz auch als Plan- bzw. Stamm-Modus bezeichnet). In engem Zusammenhang mit den beiden Programm-Zuständen stehen die zugehörigen Datei-Typen (Stammdaten- und Plandaten-Dateien), die die Persistenz-Haltung des jeweiligen *Programmzustandes* ermöglichen. Durch das Laden einer Datei eines bestimmten Typs geht das Programm also automatisch in den entsprechenden Modus über.
Der Übergang in den Plandaten-Modus erfolgt durch die Initialisierung der Planerstellung oder durch das Laden einer Plandaten-Datei. Der Wechsel vom Plan- zum Stammdaten-Zustand hat immer einen Verlust des aktuellen Planes zur Folge. Diese rigoros anmutende Programm-Eigenschaft hat natürlich Gründe. Im Plan-Modus besteht die Möglichkeit, Nebenbedingungen zu verändern, die im Stamm-Modus definiert wurden (siehe auch **Fehler! Verweisquelle konnte nicht gefunden werden.** und 3.3.3). Es wird davon ausgegangen, daß diese veränderten Restriktionen in Hinblick auf den *aktuell* bearbeiteten Plan vorgenommen wurden und daher nicht in die Menge der Nebenbedingungen der Stammdaten aufgenommen

werden sollen. Falls dies doch erwünscht ist, müssen die entsprechenden Restriktionen *vor* der Plan-Erstellung im Stamm-Modus definiert werden.
Die eben genannten Gründe und die Ausführungen des Abschnittes 3.3.3 motivieren nach jedem Wechsel zwischen Stamm- und Plan-Modus eine Entkopplung von transienten und persistenten Daten. Dies hat zur Folge, daß der erste **Speichern**-Befehl nach einem Zustands-Wechsel in ein **Speichern unter** übergeht.

4.2. Modellierung der Daten einer Schule

Wie werden die realen Strukturen einer Schule in Datenstrukturen umgesetzt, mit denen das Programm arbeitet und die teilweise auch die Bedienung des Programms beeinflussen? Wie stellen sich Unterrichts-Einheiten, Kurse u.s.w. in der Implementation dar? Was ist ein Stundenplan? Fragen dieser Art soll in diesem Abschnitt nachgegangen werden.[12]

Die visuellen Komponenten des C++ Builders verwenden nicht-visuelle Klassen für die Verwaltung ihrer Daten. Vor allem sind dies die String-Klasse `AnsiString` und die String-Liste `TStringList`. Diese Klassen zeichnen sich durch eine äußerst bequeme Handhabung aus und werden für die Modellierung der Datenstrukturen von XXX intensiv genutzt. Besonders hervorzuheben ist die Fähigkeit des String-Listen-Typs, jedem String der Liste zusätzlich ein beliebiges Objekt zuweisen zu können. Einzige Voraussetzung dabei ist, daß das Objekt von der VCL-Basisklasse `TObject` abgeleitet ist.
Die dritte, aber eigentlich wichtigste Klasse für die Programmierung, ist der Listen-Typ `TList`, welcher eine Liste von Zeigern auf Objekte verwaltet. Die Optimierungs-Algorithmen basieren auf der „Durchmischung“ großer Datenmengen. Listen sind schnell und flexibel, bzgl. des Verschiebens von Elementen innerhalb der Listen-Struktur. Daher bieten sie sich für die Modellierung der internen Datenstrukturen an.

Die Datenstrukturen lassen sich, in Anlehnung an die Programm-Zustände, in zwei Kategorien einteilen. Stammdaten und Unterrichts-Definitionen werden durch die C++ - Klassen `CLehrer`, `CKlasse`, `CEinheit`, `CFachBlock`, `CKursBlock` und `CRaumFach` repräsentiert. Die Beschreibung der Plandaten erfolgt mittels der Klassen `CPlan`, `CStunde`, `CLehrerStatus` und `CKlassenStatus`.

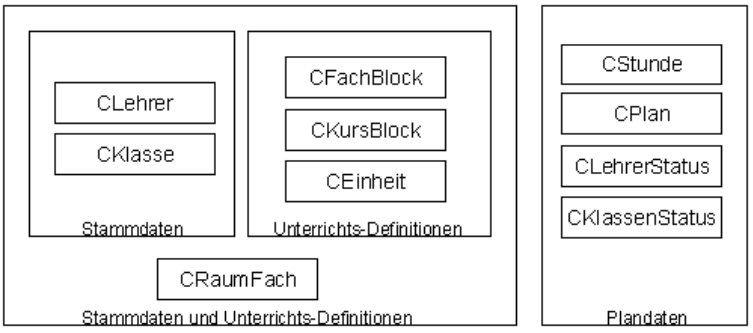


Abbildung 4: Zuordnung der C++ - Klassen zu den Programm-Zuständen

Diese Einteilung wird lediglich aus Gründen der Übersichtlichkeit vorgenommen, denn natürlich erfolgen im Plan-Zustand auch Zugriffe auf die Strukturen der Stammdaten.

[12] Alle Quelltext-Passagen, einschließlich Variablen und Typ-Bezeichner, werden im folgenden in der Schriftart `Courier New` dargestellt.

Benutzer-Schnittstellen wie z.B. Combo-Boxen verfügen über je eine Variable `Items` vom Typ `TStringList`, in welcher die Einträge der Combo-Box gespeichert werden. Für Lehrer, Klassen u.s.w. steht je eine Combo-Box zur Verfügung. Jedem String dieser Boxen ist ein Zeiger auf eine Instanz der entsprechende Klasse zugeordnet. Die Combo-Box für die Lehrer enthält also Zeiger auf Instanzen von `CLehrer`. In der Implementation der Stammdaten-Phase vereinfacht sich dadurch die Verwaltung der Menge der Lehrer und die Navigation des Benutzers in dieser Menge.

Die nun folgenden Abschnitte geben einen Überblick über Funktion und Bedeutung wichtiger Datenstrukturen und Klassen. Die Beschreibung von Standard-Methoden wie Konstruktoren, Destruktoren oder Zuweisungsoperatoren ist von nur geringem Interesse und entfällt daher.

4.2.1. Zeit-Gitter

Alle Daten, die in irgend einer Weise mit der Zeitplanung in Verbindung stehen, beziehen sich auf ein Zeit-Gitter von 7 Tagen a 13 Stunden. Dieses Gitter definiert also 91 ZEITEN oder TIME-SLOTS, die als Container für Informationen dienen. Die eindeutige Identifizierung der Zeiten wird durch eine Durchnumerierung von null bis 90 erreicht.

Das Zeit-Gitter wird durch die Angabe einer Anzahl von Tagen und Stunden durch den Programm-Nutzer auf ein rechteckiges Zeit-Fenster eingeschränkt. Wie in nebenstehender Tabelle hervorgehoben, ist die übliche Anzahl an Tagen 5 (Montag bis Freitag) und die für Stunden ca. 8.

Da die zeitliche Organisation einer Schule und damit auch die eines Stundenplanes auf der Einteilung in Tage und Stunde basiert, muß häufig eine Umrechnung zwischen der Listen-Darstellung und der Tag-Stunde-Darstellung vorgenommen werden.

Tag/ Stunde	0 (Mo)	1 (Di)	2 (Mi)	3 (Do)	4 (Fr)	5 (Sa)	6 (So)
0	0	13	26	39	52	65	78
1	1	14	27	40	53	66	79
2	2	15	28	41	54	67	80
3	3	16	29	42	55	68	81
4	4	17	30	43	56	69	82
5	5	18	21	44	57	70	83
6	6	19	32	45	58	71	84
7	7	20	33	46	59	72	85
8	8	21	34	47	60	73	86
9	9	22	35	48	61	74	87
10	10	23	36	49	62	75	88
11	11	24	37	50	63	76	89
12	12	25	38	51	64	77	90

Tabelle 4: Zeitgitter

Die Ermittlung der Zeit bei gegebenen Angaben für den Tag und die Stunde ergibt sich in folgender Weise:

```
Zeit = Tag*13 + Stunde
```

Umgekehrt ist die Nummer eines jeden time-slots Repräsentant einer Restklasse modulo 13 und die Umrechnung eines time-slots in die Tag-Stunde-Darstellung erfolgt über Integer-Division und Ermittlung des Restes bei Division durch 13:

```
Tag = Zeit / 13
Stunde = Zeit % 13
```

Die jeweils ersten Stunden eines Tages (Zeiten 0, 13, 26, ...) werden in der Software als nullte Stunden gedeutet. Sie werden allerdings nur selten von den Schulen benötigt und können, wie in obiger Tabelle dargestellt, durch den Programm-Benutzer ausgeblendet werden.

4.2.2. Lehrer

Klassen-Diagramm:

CLehrer
Name : String
Kurz : String

Faecher : TStringList* KlassenLehrer : bool Klasse : String Sperre : bool[7][13] Krank : bool Stunden : int StundenLimit : int MaxStundenNo : int ProdLimit : int Opti : bool TendenzZuFreienTagen : bool
CLehrer() CLehrer(lehrer: String, kurz: String, f: TStringList*, kll: bool, klasse: String) ~CLehrer()

Diese Klasse dient zur Aufnahme der Daten eines Lehrers.

`Name`	:	Name des Lehrers
`Kurz`	:	Kurzbezeichnung des Lehrers
`Faecher`	:	nimmt die von dem Lehrer unterrichteten Fächer auf
`KlassenLehrer`	:	zeigt an, ob der Lehrer ein Klassenlehrer ist
`Sperre`	:	nimmt für 7 Tage a 13 Stunden die Verfügbarkeiten auf
`Krank`	:	wird bei der Erstellung von Vertretungsplänen benutzt
`Stunden`	:	# der Unterrichtsstunden pro Woche
`StundenLimit`	:	angestrebte maximale # von Unterrichtsstunden pro Tag
`MaxStundenNo`	:	# Tage, die mit der Maximalzahl an Stunden belegt sein können (wird noch nicht verwendet)
`ProdLimit`	:	Maß für beste Stundenverteilung
`Opti`	:	zeigt an, ob der Lehrer in Optimierung einbezogen werden soll
`TendenzZuFreienTagen`	:	freier Tag eines Lehrers wird bevorzugt behandelt

Die boolschen Werte des Attributes `Sperre` korrespondieren mit den time-slots des in Abschnitt 4.2.1 beschriebenen Zeit-Gitters.
Für weitere Informationen zur Variablen `ProdLimit` wird auf den Abschnitt 5.4 verwiesen.

4.2.3. Klasse

Klassen-Diagramm:

CKlasse
Name : String Sperre : bool[7][13] Stunden : int StundenLimit : int ProdLimit : int StundenMax : int StundenMin : int MaxStundenNo : int Opti : bool
CKlasse() CKlasse(name: String)

Diese Klasse dient zur Aufnahme der Daten einer Klasse. Die hier nicht beschriebenen Member-Variablen haben analoge Bedeutung zur Klasse `CLehrer`.

`StundenMax`	:	max. # an Stunden an einem Tag der Woche
`StundenMin`	:	min. # an Stunden an einem Tag der Woche

4.2.4. Einheit

Klassen-Diagramm:

CEinheit	
`EStunden`	`: int`
`Zeit`	`: int`
`Raum`	`: String`
`NurUrPlan`	`: bool`
`OberesLimit`	`: bool`
`UnteresLimit`	`: bool`
`TagesLimit`	`: bool`
`Tag`	`: int`
`CEinheit(estunden:int, zeit:int)`	
`CEinheit(estunden:int, zeit:int, raum:int)`	

Einheiten dienen sowohl als Planungs-Struktur für den Nutzer in der Stammdaten-Phase, als auch zur internen Verwaltung der Plandaten. Sie ermöglichen die Definition von Mehrfachstunden und das prescheduling von Unterricht. Jeder Unterrichtsstunde ist eine Einheit übergeordnet. Beispiele für Einheiten-Definitionen werden im folgenden Abschnitt 4.2.5 gegeben.

`EStunden`	:	# der Stunden für die Einheit
`Zeit`	:	dient wahlweise als time-slot für stunden-genaues prescheduling oder als oberes/unteres Zeitlimit für die durch die Einheit definierte(n) Stunde(n)
`Raum`	:	Name eines Raumes für räumliches prescheduling, falls erwünscht
`NurUrPlan`	:	gibt an, ob zeitliches prescheduling nur für die Urplan-Erstellung gültig ist
`OberesLimit`	:	zeitliches prescheduling, Stunde(n) werden nur unterhalb des durch `Zeit` definierten time-slots gesetzt (≤ `Zeit`)
`UnteresLimit`	:	zeitliches prescheduling, Stunde(n) werden nur oberhalb des durch `Zeit` definierten time-slots gesetzt (≥ `Zeit`)
`TagesLimit`	:	Einschränkung auf einen bestimmten Tag (ein/aus)
`Tag`	:	Tag, auf den eingeschränkt wird

4.2.5. Fachblock

Klassen-Diagramm:

CFachBlock	
`Lehrer`	`: String`
`Fach`	`: String`

```
Klasse                          : String
BStunden                        : int
Einheiten                       : TList*
Trennung                        : bool
SpezialTrennung                 : int
CFachBlock()
CFachBlock(lehrer:String, klasse:String, fach:String, bstunden:int,
      einheiten:TList*, trennung:bool)
~CFachBlock()
```

Fachblöcke dienen in der Planungs-Phase als Container für die Rahmendaten „normaler“ Unterrichtsstunden bzw. FACHSTUNDEN. Fachstunden sind durch die Angaben Lehrer/ Fach/ Klasse charakterisiert, welche ihrerseits einen bestimmten Stunden-Typ definieren (siehe auch 3.2). Ein `CFachBlock`-Objekt legt den Unterricht eines bestimmten Stunden-Typs für eine Woche fest.

Lehrer	:	Lehrer, der die Stunde unterrichtet
Fach	:	Fach, welches unterrichtet wird
Klasse	:	Klasse, die unterrichtet wird
BStunden	:	# der Stunden pro Woche
Einheiten	:	Liste von `CEinheit`-Objekten
Trennung	:	gibt an, ob die Trennungsfehler optimiert werden sollen (siehe auch 3.1.3)
SpezialTrennung	:	gibt an, ob die Spezial-Trennungsfehler optimiert werden sollen (siehe auch 3.1.3)

Eine Beispiel-Instanz der Klasse `CFachBlock`:

<table>
<tr><td>Lehrer</td><td>´MEIER´</td></tr>
<tr><td>Fach</td><td>´MA´</td></tr>
<tr><td>Klasse</td><td>´9A´</td></tr>
<tr><td>BStunden</td><td>4</td></tr>
<tr><td>Einheiten</td><td>
<table>
<tr><td>Attribut/
Einheit</td><td>Einheit 0</td><td>Einheit 1</td><td>Einheit 2</td></tr>
<tr><td>EStunden</td><td>1</td><td>1</td><td>2</td></tr>
<tr><td>Zeit</td><td>4</td><td>-1</td><td>28</td></tr>
<tr><td>Raum</td><td>´´</td><td>´´</td><td>´001´</td></tr>
<tr><td>NurUrPlan</td><td>false</td><td>false</td><td>false</td></tr>
<tr><td>OberesLimit</td><td>true</td><td>false</td><td>false</td></tr>
<tr><td>UnteresLimit</td><td>false</td><td>false</td><td>false</td></tr>
<tr><td>TagesLimit</td><td>false</td><td>true</td><td>false</td></tr>
<tr><td>Tag</td><td>-1</td><td>4</td><td>-1</td></tr>
</table>
</td></tr>
<tr><td>Trennung</td><td>true</td></tr>
<tr><td>SpezialTrennung</td><td>1</td></tr>
</table>

Lehrer Meier gibt vier Stunden Mathematik pro Woche in der Klasse 9a, zwei Einzelstunden und eine Doppelstunde. Die Summe der Attribute `EStunden` der Unterrichts-Einheiten in der Liste `Einheiten` ergibt die Anzahl an Blockstunden, die im Attribut `BStunden` des `CFachBlock`-Objektes festgelegt ist. Die erste Einzelstunde findet spätestens in der 4. Stunde statt, die zweite Einzelstunde wird zu einer beliebigen Zeit am vierten Tag (i. Allg.

Freitag) gesetzt. Die Doppelstunde ist für den time-slot 28 und 29 (i. Allg. Mittwoch 2. und 3. Stunde) und den Raum 001 vorgesehen. Die Mathematikstunden sollen möglichst auf die ganze Woche verteilt werden (`Trennung`) und nicht in Kombination mit anderen Fächern des selben Wertes für `SpezialTrennung` auftreten. Alle Vorgaben gelten während des gesamten Optimierungs-Vorgangs (`NurUrPlan`).

In der formalen Problemstellung (Abschnitt 3.2) ging *jede* Definition einer Stunde aus einer Kurs-Definition hervor. Dies währe theoretisch auch implementationstechnisch möglich. Allerdings erfahren Fachstunden und Kursstunden eine unterschiedliche Behandlung in der Implementation der Stammdaten-Phase. Die Trennung von Kurs- und Fachstunden hat ihren Ursprung in der Methodik der Stunden-Definition durch den Programm-Benutzer, kann aber auch aus Implementations-Sicht als spezielle Anwendung des Prinzips divide-and-conquer verstanden werden. Nicht zuletzt würde sich ein einheitliches Modell für beide Stunden-Typen wahrscheinlich negativ auf die Lesbarkeit des Quelltextes auswirken.

Eine Alternative zu dieser Entwurfs-Entscheidung währe die Einbettung der Kurs- und Fachstunden in eine Ableitungs-Hierarchie. Die Kursstunde, als allgemeines Modell, könnte die Basisklasse der Fachstunde darstellen, die ihrerseits als Spezialisierung einer Kursstunde verstanden werden kann. Bei einem solch minimalen Ableitungs-Modell stellt sich jedoch die Frage nach der Notwendigkeit. Und allein um das Zauberwort „objektorientiert“ in die Dokumentation aufnehmen bzw. auf die Verpackung drucken zu können, sollte der zu erwartende Überbau in der Implementierung nicht in Kauf genommen werden. Dies um so mehr, als die hier beschriebenen Klassen (mit Ausnahme von `CPlan`) mehr die Funktion „besserer structs“ als die eigenständiger Typen erfüllen.

4.2.6. Kursblock

Klassen-Diagramm:

CKursBlock	
Name	: String
Kurz	: String
Faecher	: TStringList*
Lehrer	: TStringList*
Klassen	: TStringList*
Einheiten	: TList*
BStunden	: int
Trennung	: bool
SpezialTrennung	: int
CKursBlock() CKursBlock(name:String, kurz:String, faecher:TStringList*, lehrer:TStringList*, klassen:TStringList*, bstunden:int, einheiten:TList*, trennung:bool) ~CKursBlock()	

Die Struktur dieser Klasse erfaßt Unterricht in Kursstunden. Damit läßt sich, zum Teil in Verbindung mit der Klasse `CRaumFach`, eine Reihe von Spezialfällen behandeln, wie z.B. der Sport-Unterricht.
Die hier nicht erläuterten Member-Variablen haben analoge Bedeutungen wie in der Klasse `CFachBlock`.

`Name` : Bezeichnung des Kurses

`Kurz`	:	Kurzbezeichnung
`Faecher`	:	Liste von Fach-Bezeichnungen
`Lehrer`	:	Liste von Lehrern, die mit der Fach-Liste korrespondiert + zusätzlich Container für Einheiten der Kursstunden
`Klassen`	:	Liste der am Kurs beteiligten Klassen

Für eine Beispiel-Instanz der Klasse `CKursBlock` wird auf ein Beispiel aus der formalen Problemstellung zurückgegriffen:

$$s_4 = \begin{Bmatrix} \left(\{7a,7b,7c\},(Schmitt,Ch,rau(Ch)),z_\phi\right), \\ \left(\{7a,7b,7c\},(Meier,Ph,rau(Ph)),z_\phi\right), \\ \left(\{7a,7b,7c\},(Müller,Bio,rau(Bio)),z_\phi\right) \end{Bmatrix}$$

Wie in der Klasse `CFachBlock`, werden die Daten in der Implementation gegenüber dem Modell der formalen Problemstellung durch die Attribute `Einheiten`, `BStunden`, `Trennung` und `SpezialTrennung` ergänzt.

<table>
<tr><td>Name</td><td>´NATURWISSENSCHAFT_7´</td></tr>
<tr><td>Kurz</td><td>´NW7´</td></tr>
<tr><td>Faecher</td><td>´CH´, ´PH´, ´BIO´</td></tr>
<tr><td>Lehrer</td><td>´SCHMITT´, ´MEIER´, ´MÜLLER´</td></tr>
<tr><td>Einheiten</td><td>
<table>
<tr><th>Attribut/
Einheit</th><th>Einheit 0</th><th>Einheit 1</th></tr>
<tr><td>EStunden</td><td>1</td><td>1</td></tr>
<tr><td>Zeit</td><td>-1</td><td>-1</td></tr>
<tr><td>Raum</td><td>´CH1´</td><td>´CH1´</td></tr>
<tr><td>NurUrPlan</td><td>false</td><td>false</td></tr>
<tr><td>OberesLimit</td><td>false</td><td>false</td></tr>
<tr><td>UnteresLimit</td><td>false</td><td>false</td></tr>
<tr><td>TagesLimit</td><td>true</td><td>true</td></tr>
<tr><td>Tag</td><td>2</td><td>4</td></tr>
</table>
...
</td></tr>
<tr><td>Klassen</td><td>´7A´, ´7B´, ´7C´</td></tr>
<tr><td>BStunden</td><td>2</td></tr>
<tr><td>Trennung</td><td>true</td></tr>
<tr><td>SpezialTrennung</td><td>1</td></tr>
</table>

Lehrer und Fächer eines Kursblockes sind entsprechend ihrer Position in den Listen einander zugeordnet. Die Anzahl der Lehrer bzw. die der Fächer legt die Anzahl der zeitparallel stattfindenden Stunden fest. Die am Kurs beteiligten Klassen verteilen sich auf die verfügbaren Stunden. Da es für jede einzelne der Stunden möglich sein sollte, den Raum vorzudefinieren, wird für jede Stunde ein `CEinheit` - Objekt benötigt. Jedem String in `Lehrer` wird ein Zeiger auf eine Instanze von `CEinheit` zugewiesen. In die obigen Darstellung wurde aus Platzgründen nur die Einheit der Chemie-Stunde bei Lehrer SCHMITT aufgenommen.

Da die Kursstunden zeitparallel stattfinden, liegen die Daten, die sich auf das zeitliche prescheduling beziehen, in mehrfacher Ausführung vor. Für das Beispiel bedeutet dies, daß die insgesamt sechs Stunden Mittwochs bzw. Freitags (Tag 2 bzw. 4) stattfinden.
Diese Redundanz von Daten wird in Kauf genommen, um keinen neuen Typ von Einheiten definieren zu müssen. Die Existenz der Variable `Einheiten` in `CKursBlock` ist nun eigentlich nicht mehr nötig. `Einheiten` übernimmt zwar in vielen Situationen die Rolle einer temporären Variablen. Trotzdem ist diese Implementierung nicht sonderlich elegant und läßt sich nur durch die Umschreibung „historisch entstanden“ rechtfertigen[13].
Die Eigenschaft `Trennung` bezieht sich auf die weiteren Stunden des Kurses und nicht auf das eigentlich unterrichtete Fach. Dies läßt sich nur schwerlich anders realisieren, da nicht überprüft werden kann, welche Schüler aus welcher Klasse welche Kursstunden gewählt haben. Allerdings können über die Eigenschaft `SpezialTrennung` Kurs- und Fachstunden kombiniert werden. Betrachtet man die Beispiel-Instanzen für `CFachBlock` und `CKursBlock` im Kontext eines gemeinsamen Planungs-Vorganges, führt der übereinstimmende Wert der Variablen `SpezialTrennung` zu dem Versuch, möglichst selten zwei der insgesamt sechs Stunden pro Woche an einem Tag stattfinden zu lassen. Rein rechnerisch ist eine Trennung bei fünf Schultagen natürlich nicht möglich. Die beste Verteilung der Stunden auf fünf Tage wäre also 4 × 1 und 1 × 2.

4.2.7. Raum

Klassen-Diagramm:

CRaumFach

`Name : String`
`Kurz : String`
`Faecher : TStringList*`
`Universal : bool`
`Spezial : bool`
`SpezialReserviert : bool`
`KlassenKursFreigabe : bool`
`Klasse : String`
`KeinKlassenRaum : bool`
`KlassenRaumReserviert : bool`
`Sperre : bool[7][13]`

`CRaumFach()`
`CRaumFach(name:String, kurz:String, f:TStringList*, u:bool, s:bool, sr:bool, kl:String, keinKRaum:bool, kRaum:bool, nurKlasse:bool, klKurs:bool)`
`~CRaumFach()`

Diese Klasse erfüllt verschiedene Aufgaben. Es werden die Räume an sich und die Zuordnungen zwischen Fächern und Räumen verwaltet, womit das räumliche prescheduling unterstützt wird. Bestimmte Fächer (Spezial-Fächer) müssen in speziell dafür vorgesehenen Räumen stattfinden, wie z.B. Chemie oder Sport. Die Klasse ermöglicht eine explizite *Zuordnung von Fächern zu Räumen* und nicht umgekehrt. Es ist also nicht möglich, für ein Fach explizit eine Menge von Räumen anzugeben, in welcher das Fach unterrichtet werden kann. Diese Information muß aus der Menge aller `CRaumFach`-Objekte extrahiert werden,

[13] Genau genommen liegt dieser Implementierung eine spät vorgenommene Veränderung des Lösungsmodells zu Grunde, deren Auswirkungen hinsichtlich des Umschreibens umfangreicher Abschnitte des Quelltextes auf diese Weise eingeschränkt werden konnten - also eine Art „ökonomische“ Entwurfsentscheidung.

indem für jeden Raum überprüft wird, ob dieser das entsprechende Fach erlaubt. Währe die umgekehrte Zuordnung möglich, müßte für Fächer, die keine Spezial-Fächer sind, stets die Menge *aller* möglichen Räume angegeben werden.
Weiterhin können mit Hilfe der Klasse `CRaumFach` Räume als Klassenräume festgelegt werden.

`Name`	:	Bezeichnung des Raumes
`Kurz`	:	Kurzbezeichnung des Raumes
`Faecher`	:	Liste der Fächer, die in diesem Raum unterrichtet werden können
`Universal`	:	zeigt an, ob der Raum ein Universal-Raum ist
`Spezial`	:	zeigt an, ob der Raum ein Spezial-Raum ist
`SpezialReserviert`	:	zeigt an, ob der Raum ein reservierter Spezial-Raum ist
`KlassenKursFreigabe`	:	Kursstunden mit Beteiligung der Klasse dieses Raumes dürfen in diesem Raum stattfinden
`Klasse`	:	Klassen-Name, falls der Raum ein Klassenraum ist
`KeinKlassenRaum`	:	zeigt an, ob der Raum kein Klassenraum ist
`KlassenRaumReserviert`	:	zeigt an, ob der Raum ein reservierter Klassenraum ist
`Sperre`	:	nimmt die Verfügbarkeiten für den Raum auf

Die Felder `Universal`, `Spezial` und `SpezialReserviert` beziehen sich auf Fächer, während `Klasse`, `KeinKlassenRaum`, `KlassenRaumReserviert` und `KlassenKursFreigabe` mit der Definition von Klassenräumen in Verbindung stehen.
Von den drei ersteren Variablen kann immer nur genau eine mit dem Wert `true` belegt sein. Gleiches gilt für `KeinKlassenRaum` und `KlassenRaumReserviert`. Nur wenn das Feld `KlassenRaumReserviert` den Wert `true` besitzt, haben `KlassenKursFreigabe` und `Klasse` eine Bedeutung. `KlassenRaumReserviert` ist also eine Art „Hauptschalter“ für Klassen-Raum-Definitionen.

Drei Beispiel-Instanzen der Klasse `CRaumFach`:

Name	´PHYSIK1´		´001´		´002´
Kurz	´PH1´		´001´		´002´
Faecher	´PH´		´´		´´
Universal	false		true		true
Spezial	true		false		false
SpezialReserviert	false		false		false
KlassenKursFreigabe	false		true		false
Klasse	´´		´10A´		´´
KeinKlassenRaum	false		false		false
KlassenRaum	false		false		false
KlassenRaumReserviert	false		true		false
Sperre	...		...		...

Die Räume 001 und 002 sind Universal-Räume (`Universal == true`), während der Raum PH1 ein Spezial-Raum ist. Alle Fächer, die in keiner Fächer-Liste der `CRaumFach`-Objekte zu finden sind, können damit in den Räumen 001 und 002 unterrichtet werden. Enthalten Variablen das Wort „Spezial“ bedeuten dies, daß die Fächer der String-Liste `Faecher` nur in bestimmten Räumen stattfinden können. Das Fach PH kann nur im Physik-Raum unterrichtet werden, weshalb auch die Variable `Spezial` mit dem Wert `true` belegt ist. Allerdings können auch andere Fächer gegeben werden, etwa Deutsch im Physikraum.
Die Daten des ersten Beispiels haben zur Folge, daß alle Physik-Stunden im Raum PHYSIK1 stattfinden und daß, falls möglich, noch andere Stunden in diesen Raum plaziert werden können. Falls mehr als ein Physik-Raum definiert wird, teilen sich die Physik-Stunden auf dieses Räume auf. Es währe auch möglich, dem Raum PHYSIK1 weitere Fächer zuzuweisen - z.B. Chemie. Dann währe dieser Raum für die Spezial-Fächer Physik und Chemie und für alle Normal-Fächer geeignet. Wird die Variable `SpezialReserviert` auf true gesetzt (anstelle von `Spezial`), können keine Normal-Fächer mehr in diesem Raum unterrichtet werden. Dies ist in der Praxis der Normalfall, da vor allem die Spezial-Räume Engpässe in der Planung darstellen und es nicht wünschenswert ist, den spärlichen Platz zu verschwenden.
Die etwas unübersichtlich erscheinende Programm-Logik der Räume ist auf der Benutzer-Ebene mit Hilfe komfortabler Bedienungs-Elemente weitgehend transparent umgesetzt. Traditionell etwas schwieriger gestaltet sich jedoch die Definition bestimmter Sonderfälle - speziell der Sport-Stunden. Turnhallen sind quasi die Härtefälle unter den Spezial-Räumen. Eine Turnhalle ist ein Raum, in welchem Unterricht oft in verschiedensten Konstellationen stattfindet:

- ein Lehrer + eine Klasse
- ein Lehrer + zwei Klassen
- zwei Lehrer + eine Klasse
- zwei Lehrer + zwei Klassen
- ...

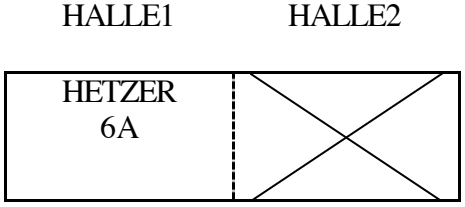
... mit anderen Worten: x Lehrer + y Klassen.

Mit dem vorliegenden Modell lassen sich solche Konstellationen mit „Pseudo-Räumen“ realisieren - ein Beispiel: Klasse 10A hat den Raum HALLE für sich allein, wird in Jungen und Mädchen unterteilt und hat bei den Lehrern HETZER und ANATOM Sport. Die Klassen 7A und 7B haben gemeinsam in der Turnhalle Sport - allerdings separat unterrichtet durch die beiden Lehrer. Das Problem besteht nun darin, daß die Stunde der 10A nur einen, die Stunden der 7A und 7B, die zeitlich parallel stattfinden, aber eigentlich zwei Räume benötigen, obwohl tatsächlich nur einer zur Verfügung steht. Oder anders formuliert: das Modell von XXX basiert auf der Annahme, daß die Belegung eines Raumes durch eine Unterrichtsstunde stets von der Zuweisung nur *eines* Lehrers an diese Stunde und damit an diesen Raum begleitet ist.
Dies erfordert die Einführung eines weiteren Raumes HALLE2, dem das Spezial-Fach Sport zugewiesen werden muß. Die folgende Darstellung veranschaulicht die nun entstandene Situation:

HALLE1	HALLE2
HETZER 10A(J)	ANATOM 10A(M)

HALLE1	HALLE2
HETZER 7A	ANATOM 7B

Für die Definition dieses Unterrichts müssen in jedem Fall Kurse definiert werden. So ist die Stunde von 7A und 7B z.B. der Typ einer klassischen Kursstunde.
Falls nun an der selben Schule noch die Konstellation ein Lehrer + eine Klasse für die Turnhalle eintritt muß sichergestellt werden, daß dieser Sport-Unterricht nur in *eine* der jetzt zwei Turnhallen plaziert wird, da in diesem Moment theoretisch noch die jeweils andere Halle für Sport-Stunden diesen Typ vergeben werden könnte. Dies wird erreicht, indem allen Einheiten der Sport-Stunden mit nur einem Lehrer und einer Klasse explizit eine der beiden Hallen als Raum zugewiesen wird (Variable `Raum` von `CEinheit`). Die Definition dieser Stunden kann auf einfache Weise mit Hilfe der Klasse `CFachBlock` erfolgen.



HALLE2 ist nun für Sport-Stunden mit nur einem Lehrer und einer Klasse nicht mehr verfügbar.

4.2.8. Stunde

Klassen-Diagramm:

CStunde	
Lehrer	: int
Klasse	: int
FachKurs	: int
KursKlassenL	: TList*
RaumL	: TList*
RaumLIndex	: int
KursStundenL	: TList*
KursStundenLIndex	: int
EStunden	: int
Zeit	: int
BlockPos	: int
Fest	: int
Trennung	: bool
SpezialTrennung	: int
NurUrPlan	: bool
OberesLimit	: bool
UnteresLimit	: bool
TagesLimit	: bool
Tag	: int
CStunde()	
CStunde(stunde:const CStunde&)	
~CStunde()	

Diese Klasse ist die Informations-Einheit, aus der die eigentlichen Stundenpläne bestehen. `CStunde`-Objekte gehen aus den Unterrichts-Definitionen in den Instanzen der Klassen `CFachBlock` und `CKursBlock` hervor bzw. dienen als Leerstunden in den Bereichen der Stundenpläne, die nicht von Unterricht besetzt sind. Sie basieren in ihrer Struktur auf dem Begriff der Stunden-Definition aus der formalen Problemstellung.

Da die Optimierungs-Algorithmen intensiv mit der Klasse `CStunde` arbeiten, wird hier vollständig zu einer Integer-Repräsentation von Lehrern, Klassen u.s.w. übergegangen[14].
Ein weiterer Tribut an die Forderung nach Geschwindigkeit ist die Verwendung der Klasse nicht als Datentyp, sondern als Datenstruktur. Programmier-Konzepte wie das Kapseln interner Datenstrukturen durch Operationen (Geheimnis-Prinzip) werden nicht angewandt.
Die Vermeidung von Funktions-Aufrufen in der Implementierung von Optimierungs-Verfahren ist eine gängige Maßgabe, um schlanke und damit schnelle Algorithmen zu erhalten.
Auch hier stellt sich die Frage, ob die beiden Typen von Unterrichts-Definitionen (`CFachBlock` und `CKursBlock`) ebenso durch unterschiedliche Stunden-Objekte in den Plan-Strukturen repräsentiert werden sollten. Die gemeinsame Modellierung von Kurs- und Fachstunden in der Klasse `CStunde` ist letztlich eine Entwurfs-Entscheidung und die Vor- und Nachteile einer Ableitungshierarchie als Alternative (siehe auch 4.2.5 Fachblock) sind nur schwer abzuschätzen.

`Lehrer`	:	Lehrer-ID
`Klasse`	:	Klassen-ID (-1 ⇒ Stunde ist eine Kursstunde)
`FachKurs`	:	Kurs-ID oder Fach-ID
`KursKlassenL`	:	Liste von Klassen (Zeiger auf ID), die an dem Kurs beteiligt sind
`RaumL`	:	Liste der Räume (Zeiger auf ID, in welche die Stunde gesetzt werden kann
`RaumLIndex`	:	Position des Listen-Zeigers in `CPlan::Alle` (siehe 4.2.11)
`KursStundenL`	:	Liste von Zeigern auf alle zeitparallelen Stunden (einschließlich der Stunde selbst)
`EStunden`	:	Länge der Unterrichts-Einheit (> 1 ⇒ Mehrfachstunde)
`Zeit`	:	zeitliches prescheduling, wenn ≠ -1
`BlockPos`	:	Position der Stunde innerhalb einer Mehrfach-Einheit
`Fest`	:	zeigt den prescheduling-Zustand an; mögliche Zustände sind: nicht fest, fest in Zeit, fest in Raum, fest in Raum und Zeit

Die letzten sieben hier nicht aufgezählten Variablen (`Trennung`, `SpezialTrennung`, ...) haben die selbe Bedeutung wie in den Klassen der Fach- und Kurs-Blöcke.

Da `CStunde` sowohl Kurs- als auch Fachstunden modelliert, kann es vorkommen, daß je nach Stunden-Art bestimmte Variablen ungenutzt sind. Konkret betrifft das `KursKlassenL` und `KursStundenL`, die im Fall einer Fachstunde keine Daten enthalten. Im Gegensatz dazu hat das Feld `FachKurs` praktisch zwei Bedeutungen, da sich die Wertebereiche für Fach-ID's und Kurs-ID's überschneiden - daher die Wahl für den Variablen-Bezeichner.
Der Fall `Klasse == -1` kennzeichnet die Stunde als Kursstunde. In diesem Fall enthalten die Listen `KursKlassenL` und `KursStundenL` Informationen führt die Kopplung der zeitparallelen Kursstunden.
Da an eine Unterrichts-Stunde stets genau ein Lehrer gebunden ist und der Wertebereich der Lehrer-ID's mit 0 beginnt, dient die Variable `Lehrer` genau dann als Kennzeichnung für eine unbelegte Stunde, wenn sie mit dem Wert -1 belegt ist.
Die Liste `RaumL` enthält die ID's aller für die jeweilige Stunde verfügbaren Räume. Insbesondere bedeutet dies, daß alle anderen Räume für diese Stunde unsichtbar sind. Diese

[14] Diese Zahlen (beginnend bei 0) werden im folgenden auch mit ID (Identifikation) bezeichnet: Lehrer-ID, Klassen-ID u.s.w.

Einschränkung auf bestimmte Räume spielt eine große Rolle bei der Steigerung der Effizienz der Optimierungs-Algorithmen. Aber abgesehen von der Erhöhung der Rechengeschwindigkeit ist diese Einschränkung natürlich auch ein Bestandteil des eigentlichen Problems - nämlich der Frage, welche Fächer oder welche Klassen in welche Räumen gesetzt werden dürfen. Die Erzeugung der Liste `RaumL` erfolgt also auf der Grundlage der Informationen in den `CRaumFach`-Objekten. Häufig steht für eine Anzahl von Stunden die selbe Menge von Räumen zur Verfügung. Vor allem sind dies Normalstunden, deren Klassen keinen Klassenraum besitzen, Stunden also, die in allen Normal-Räumen stattfinden können. Die entsprechenden `CStunde`-Objekte, teilen sich *eine* Liste von Raum-ID's, die in der Member-Variablen `CPlan::UniversalL` abgelegt ist (4.2.11). Damit wird zwar kein Geschwindigkeits-Vorteil erzielt, aber der Speicherbedarf des Programmes wird herabgesetzt.
Die Variable `Fest` schützt die Stunde während der Optimierung vor ungültigen zeitlichen oder räumliche Verschiebungen. Ferner dient sie dazu, nachträgliche Veränderungen des preschedulings durch den Plan-Ersteller zu ermöglichen.
Die Variable `RaumLIndex` wird für die Speicherung der Plan-Daten benötigt.
Auf ein Beispiel für eine Instanz der Klasse `CStunde` wird hier verzichtet, da die Integer-Darstellung der Member-Variablen nicht sonderlich aussagekräftig ist.

Weitere Ausführungen zur Verwendung der Klasse `CStunde` werden im Zusammenhang mit der Klasse `CPlan` (4.2.11) und deren Methoden vorgenommen.

4.2.9. Klassenstatus

Klassen-Diagramm:

CKlassenStatus	
`Klasse`	`: int`
`LeerStunden`	`: int`
`LeerStundenMax`	`: int`
`StundenMax`	`: int`
`StundenMin`	`: int`
`StundenProd`	`: int`
`TrennungsFehler`	`: int`
`SpezialTrennungsFehler`	`: int`
`Opti`	`: bool`
`CKlassenStatus()`	
`operator =(klasse:const CKlassenStatus&) : CKlassenStatus&`	

Diese Datenstruktur faßt den Zustand einer Klasse bzgl. der Erfüllung der Restriktionen während der Plan-Optimierung zusammen.

`Klasse`	:	Lehrer-ID
`LeerStunden`	:	# der Leerstunden
`LeerStundenMax`	:	Maximum der Leerstunden an einem Tag
`StundenMax`	:	Maximum der Unterrichtsstunden an einem Tag
`StundenMin`	:	Minimum der Unterrichtsstunden an einem Tag
`StundenProd`	:	Maß für die Stundenverteilung
`TrennungsFehler`	:	# der Trennungsfehler
`SpezialTrennungsFehler`	:	# der Spezial-Trennungsfehler
`Opti`	:	zeigt an, ob der Lehrer optimiert werden soll

Die Berechnung der Member-Variablen wird in der Methode `CPlan::KStatus` vorgenommen.
Mit der Variable `Opti` läßt sich eine Klasse vollständig aus dem Optimierungs-Vorgang ausblenden. Dies ist von Vorteil, wenn z.B. der Gebrauch von Pseudo-Klassen erwünscht ist[15].

4.2.10. Lehrerstatus

Klassen-Diagramm:

CLehrerStatus	
Lehrer	: int
LeerStunden	: int
LeerStundenMax	: int
StundenMax	: int
StundenMin	: int
StundenProd	: int
Opti	: bool
CLehrerStatus()	
operator =(lehrer:const CLehrerStatus&) : CLehrerStatus&	

Diese Klasse faßt den Zustand eines Lehrers bzgl. der Erfüllung der Restriktionen während der Plan-Optimierung zusammen. Die Bedeutungen der Variablen entsprechen denen der Klasse `CKlassenStatus`. Das Optimierungs-Ziel der Lehrer-Leerstunden wird im folgenden mit LLS bezeichnet.
Die Ausblendung eines Lehrers aus dem Optimierungs-Vorgang mittels der Variable `Opti` ist bei bestimmten Sonderfällen vorteilhaft[16].

4.2.11. Plan

Klassen-Diagramm:

CPlan	
RL	: TList*
LehrerL	: TList*
KlassenL	: TList*
RaumL	: TList*
Stunden	: int
Tage	: int
Nullte	: bool
L	: int
K	: int
R	: int
LLSSum	: int

[15] Als Beispiel möge hier die Definition von Bereitschaftsstunden für alle Lehrer dienen, die die Erzeugung einer Pseudo-Klasse und eines oder mehrerer Pseudo-Räume vorraussetzt. In diesem Fall ist es nicht erforderlich, die Leerstunden dieser Klasse zu beseitigen.
[16] Dies sind z.B. Lehrer, die wegen eines seltenen Faches (Religion, Latein, ...) oder zwecks Betreuung lernbehinderter Kinder in mehreren verschiedenen Schulen unterrichten. Darüber hinaus stehen bei solchen Lehrkräften häufig die Unterrichts-Zeiten von vornherein fest, so daß eine Optimierung so oder so keinen Sinn machen würde. Ein weiteres Beispiel sind Lehrkräfte im Vorruhestand oder Lehrer, die in der Schulleitung tätig sind und die aus diesem Grund nur wenige Stunden pro Woche unterrichten.

```
SLLSSum                         : int
KLSSum                          : int
TFSum                           : int
TFMax                           : int
SpezialTFSum                    : int
SpezialTFMax                    : int
LLSMax                          : int
LLSLimit                        : int
LProdSum                        : int
KProdSum                        : int
SKProdSum                       : int
T                               : float
LProdSumLimit                   : int
KProdSumLimit                   : int
KursUeberStundenLimit           : bool
MehrfachKursUeberStundenLimit : bool
MehrfachUeberStundenLimit     : bool
AbbruchErsteStunde              : bool
Init                            : bool
LStatusL                        : TList*
KStatusL                        : TList*
Alle                            : TList*
UniversalL                      : TList*
LeerStunde                      : CStunde*
ProblemStunde                   : CStunde*
ProblemRaum                     : int
ProblemKlasse                   : int
ProblemLehrer                   : int
ProblemTyp                      : int
Probs                           : const char[10]
```

```
CPlan()
~CPlan()
CPlan(stunden:int, tage:int, nullte:bool,lehrerL:TList*,   klassenL:
      TList*, raumL:TList*)
CPlan(P:const CPlan&)
SetzTest(stunde: CStunde*, zeit: int, raum: int) : CStunde*
LStatus(LIndex:int): void
KStatus(KIndex:int) : void
InitUrPlan() : int
ReInit() : int
GlobalParas() : void
ParaInit() : void
KLSMinimierung(typ:int, SuchNo:int) : void
Austausch(typ:int,klassenLimit:int,tausch:int) : int
KlassenTest(kl:int) : int
Tausch(klasse1:int, klasse2:int, zeit1:int, zeit2:int, pS1:CStunde*,
      pS2:CStunde*, DoIt:bool, manuell:bool, OptTyp:const char) : bool
F_T() : float
```

Dies ist die zentrale Datenstruktur in der Phase der Plan-Erstellung.

`RL`	:	Liste von Listen von `CStunde` - Objekten
`LehrerL`	:	Liste von Kopien der `CLehrer`-Objekte aus der Stammdaten-Phase
`KlassenL`	:	Liste von Kopien der `CKlasse`-Objekte aus der Stammdaten-Phase
`RaumL`	:	Liste von Kopien der `CRaumFach`-Objekte aus der Stammdaten-Phase
`Stunden`	:	maximale Stundenzahl pro Tag

`Tage`	:	Anzahl von Tagen pro Woche
`Nullte`	:	Verwendung der nullten Stunde
`L`	:	Anzahl der Lehrer
`K`	:	Anzahl der Klassen
`R`	:	Anzahl der Räume
`LLSSum`	:	Summe der Leerstunden aller Lehrer
`SLLSSum`	:	Parameter für Sintflut-Optimierung der Lehrer-Leerstunden
`KLSSum`	:	Summe der Leerstunden aller Klassen
`TFSum`	:	Summe der Trennungsfehler aller Klassen
`TFMax`	:	Maximum der Trennungsfehler aller Klassen
`SpezialTFSum`	:	Summe der Spezial-Trennungsfehler aller Klassen
`SpezialTFMax`	:	Maximum der Spezial-Trennungsfehler aller Klassen
`LLSMax`	:	Maximum der Leerstunden aller Lehrer
`LLSLimit`	:	angestrebtes Maximum an Lehrer-Leerstunden für alle Lehrer
`LProdSum`	:	Maß für die Stundenverteilung der Stunden der Lehrer
`KProdSum`	:	Maß für die Stundenverteilung der Stunden der Klassen
`SKProdSum`	:	Parameter für Optimierung der Stundenverteilung der Klassen durch Sintflut-Verfahren bzw. Simulated Annealing
`T`	:	Temperatur für Simulated Annealing bzw. Akzeptanz-Intervall beim Sintflut-Verfahren
`LProdSumLimit`	:	optimaler Wert für das Maß der Stundenverteilung der Stunden der Lehrer
`KProdSumLimit`	:	optimaler Wert für das Maß der Stundenverteilung der Stunden der Klassen
`KursUeberStundenLimit`	:	Dürfen Kurse prinzipiell über dem Stunden-Limit gesetzt werden?
`MehrfachKursUeberStundenLimit`	:	Dürfen Mehrfach-Kurse prinzipiell über dem Stunden-Limit gesetzt werden?
`MehrfachUeberStundenLimit`	:	Dürfen Mehrfachstunden prinzipiell über dem Stunden-Limit gesetzt werden
`AbbruchErsteStunde`	:	zeigt an, ob die Urplan-Erstellung als gescheitert betrachtet werden soll, wenn nicht alle Klassen mit der ersten Stunde den Tag beginnen
`Init`	:	Zustandsvariable: kennzeichnet unvollständigen Zustand der Raum-Listen (z.B. während Urplan-Erstellung)
`LStatusL`	:	Liste der `CLehrerStatus`-Objekte
`KStatusL`	:	Liste der `CKlassenStatus`-Objekte
`Alle`	:	Liste aller nicht leeren `CStunde`-Objekte
`UniversalL`	:	Liste der Universal-Räume (siehe 4.2.8 Stunde)
`LeerStunde`	:	Hilfsvariable (siehe Abschnitt 4.4.2)
`ProblemStunde`	:	wird für die Fehlerdiagnose durch den Benutzer mit Informationen zum aufgetretenen Problem belegt
`ProblemRaum`	:	ID des Problem-Raums
`ProblemKlasse`	:	ID der Problem-Klasse
`ProblemLehrer`	:	ID des Problem-Lehrers
`ProblemTyp`	:	definiert den Typ des aufgetretenen Problems
`Probs`	:	verknüpft Problem-Typ mit beschreibender Zeichenkette

Dieser Abschnitt beschäftigt sich vorrangig mit den Datenstrukturen eines Plan-Objektes. Detailliertere Beschreibungen einiger Methoden erfolgen im Abschnitt 5. Da bestimmte Member-Variablen nur im Kontext der Optimierungs-Algorithmen zu verstehen sind, erfolgt deren Beschreibung ebenfalls im Abschnitt 5.

Die grundlegenden Datenstrukturen innerhalb einer `CPlan`-Instanz sind die Variablen `RL` und `Alle`. Die Liste `Alle` enthält alle Unterrichtsstunden, die in einer Woche an einer Schule gegeben werden sollen. Beim Übergang in die Planungs-Phase wird diese Liste ein einziges mal aufgebaut und bei jeder Erzeugung eines neue Planes stets wiederverwendet. Erst beim Wechsel in die Stammdaten-Phase werden die `CStunde`-Instanzen, auf die die Elemente der Liste zeigen, vernichtet.
Jede in der Liste `RL` wiederum enthaltene Liste setzt sich aus den in 4.2.1 beschriebenen 91 time-slots für Zeiger auf `CStunde`-Objekte zusammen. Die Raum-Listen enthalten Kopien der Zeiger aus `Alle`[17].
Die Stunden in `Alle` sind in einer bestimmten wohl definierten Reihenfolge angeordnet. Dadurch wird das Auffinden von aneinander gekoppelten Stunden (Kurs- und Mehrfachstunden) möglich bzw. vereinfacht.
Die Variablen `Stunde`, `Tag` und `Nullte` definieren das Zeit-Fenster, auf welches die Planerstellung eingeschränkt wird. Diese Einschränkung basiert letztlich auf einer reinen Effizienz-Überlegung, da nun bei Suchvorgängen innerhalb der Raum-Listen nicht stets über alle 91 time-slots eines Raumes iteriert werden muß. Eine Iteration über das Zeit-Fenster erfordert allerdings die Umrechnung der Listen- in die Tag-Stunde-Darstellung (4.2.1).

Während der Plan-Erstellung bleibt die Anzahl von Lehrern, Klassen und Räumen konstant. Aus diesem Grund wurden die Variablen `L`, `K` und `R` eingeführt. Durch diese Variablen wird an vielen Stellen der Zugriff auf Verwaltungs-Funktionen der Listen `LehrerL`, `KlassenL` und `RaumL` vermieden. Dies wirkt sich zwar nur geringfügig positiv auf die Geschwindigkeit aus. Es ist jedoch anzunehmen, daß sich auch kleine Zeitersparnisse über einen langen Zeitraum hin bemerkbar aufsummieren.
Die eben genannten drei Listen sind Kopien von Stammdaten. Rein programmtechnisch währe dies nicht nötig, da ein direkter Zugriff auf die Stammdaten jederzeit möglich ist. Allerdings stellt ein `CPlan`-Objekt auf diese Weise eine von der Stammdaten-Phase unabhängigere Struktur dar[18].

Während erster Tests der Optimierungs-Algorithmen zeigte sich, daß sich die Positionierung von Kursen und Mehrfachstunden an Zeiten über dem Stunden-Limit (siehe `CKlasse::StundenLimit`) negativ auf die Erreichung der Optimierungs-Ziele auswirkt. Daher wurden die Variablen `KursUeberStundenLimit`, `MehrfachKursUeberStundenLimit` und `MehrfachUeberStundenLimit` eingeführt. Mit diesen Variablen kann das Setzen von Kursen und Mehrfachstunden unterhalb

[17] Korrekterweise müßten die Elemente der Raum-Listen stets als „Zeiger auf `CStunde`-Objekte“ bezeichnet werden. Im Interesse besserer Lesbarkeit des Textes wird jedoch häufig von „`CStunde`-Objekten“ oder einfach nur von „Stunden“ gesprochen. Da der Speicherplatz von VCL-Objekten stets dynamisch alloziert wird, gilt diese sprachliche Ungenauigkeit für viele Variablen bzw. Objekte.

[18] Dies kann auch als eine spekulative Entwurfs-Entscheidung mit Blick in die Zukunft verstanden werden. Sie zielt auf die gleichzeitige Verwaltung mehrerer `CPlan`-Instanzen nach dem Vorbild einer Population in einem genetischen Algorithmus ab oder auf die Erzeugung einer Anzahl von autonomen Plänen, die als verschiedene Lösungswege aus einem einzelnen Plan hervorgehen.

des Stunden-Limits erzwungen werden. Sie stellen also eine zusätzliche Form des zeitlichen preschedulings dar.

4.2.11.1. Die Methode `CPlan::SetzTest`

Bezeichner	`CPlan::SetzTest`

Parameter	`CStunde* stunde`
	`int zeit`
	`int raum`

Rückgabe-Typ	`CStunde*`
Kurz-Beschreibung	– Überprüfung, ob ein Setzen der Stunde `stunde` zur Zeit `zeit` in den Raum `raum` die Gültigkeit des Planes erhält – folgende Überprüfungen werden vorgenommen: • Überschneidungen bei Lehrern, Klassen und Räumen • Verfügbarkeiten bei Lehrern, Klassen und Räumen • zeitliches prescheduling der Stunde – Rückgabe von 0, wenn Stunde setzbar – Rückgabe ungleich 0 bei Konflikten und Speicherung der Konflikt-Informationen in den `CPlan`-Variablen `ProblemStunde`, `ProblemRaum`, `ProblemLehrer`, `ProblemKlasse` und `ProblemTyp`

Sie ist eine der wichtigsten und am häufigsten benutzten Methoden von `CPlan`. Jedem Verschieben von `CStunde`-Objekten in den Raum-Listen geht ein Aufruf der Funktion `SetzTest` voraus. Auch bei der Erstellung des Urplanes wird diese Funktion benutzt (siehe auch 5.2).
Die Funktion nimmt keine Änderungen der Plan-Struktur vor.

Bei der Urplan-Erstellung und der manuellen Veränderung bestehender Stundenpläne werden häufig Informationen benötigt, anhand derer die Ursachen für eine Reihe von Konflikt-Konstellationen für den Benutzer verdeutlicht werden können. Versucht der Benutzer z.B. manuell eine beliebige Stunde zu verschieben und führt diese Verschiebung zu einer zeitlichen Überschneidung, sollte er über die Ursache dieses Konfliktes in Kenntnis gesetzt werden können. Realisiert wird diese Programm-Eigenschaft durch die Funktion `SetzTest` und die Variablen, deren Bezeichner mit dem Wort „`Problem`“ beginnen. Es wurden sieben `char`-Konstanten definiert, die im Konflikt-Fall in der Variablen `ProblemTyp` abgelegt werden und anhand derer die Art des Konfliktes identifiziert werden kann:

Konstante	**Art des Konfliktes**
`P_RAUM`	Der zu belegende Raum ist zu der Zeit bereits belegt.
`P_KLASSE`	Die Klasse ist zu dieser Zeit bereits vergeben.
`P_LEHRER`	Der Lehrer ist zu dieser Zeit bereits vergeben.
`P_LIMIT`	Die Zeit kollidiert mit dem prescheduling eines zeitlichen Limits.
`P_SPERRUNG`	Der Lehrer/ die Klasse/ der Raum ist zu dieser Zeit gesperrt.
`P_ZEIT`	Die Zeit kollidiert mit einem expliziten zeitlichen prescheduling.
`P_ERSTE`	wird in Funktion `CPlan::InitUrPlan` benutzt

Tabelle 5: Konflikt-Konstanten

In Abhängigkeit von dem Konflikt-Typ werden in den übrigen Variablen weitere Informationen zu dem Konflikt abgelegt. Als Beispiel folgt ein kleiner Auszug aus dem Quelltext der Funktion `SetzTest`:

```
...
1     CStunde *pS;
2     pS = (CStunde*)(((TList*)(RL->Items[raum]))->Items[zeit]);
3     if ((pS->Lehrer != -1))
4     {
5           ProblemStunde = pS;
6           ProblemRaum = raum;
7           ProblemTyp = P_RAUM;
8           return pS;   //Raum ist belegt
9     }
...
```

In Zeile 2 wird der Inhalt des time-slots des betreffenden Raumes ermittelt. Mit der Operation `Items` kann also eine bestimmte Speicherstelle einer Liste referenziert werden. Ist für diese Stunde ein Lehrer eingetragen (Zeile 3), bedeutet dies, daß der Raum zu der gegebenen Zeit bereits belegt ist. Damit liegt ein Raum-Konflikt vor (Zeile 7) und als Informationen für die Fehlerdiagnose werden die ID des Raumes selbst (Zeile 6) und die Konflikt-Stunde (Zeile 5) vermerkt.

4.2.11.2. Die Methode `CPlan::ReInit`

Bezeichner	`CPlan::ReInit`
Parameter	
Rückgabe-Typ	`void`
Kurz-Beschreibung	– füllt alle Raum-Listen mit leeren `CStunde`-Ojekten auf

Vor jeder neuen Urplan-Erstellung wird diese Funktion ausgeführt. Da die Zeiger aller nicht leeren `CStunde`-Objekte in der Variablen `Alle` enthalten sind, können einfach die belegten Positionen der Raum-Listen durch neue (leere) `CStunde`-Objekte ersetzt werden, um die Erzeugung eines neuen Planes zu ermöglichen.

4.3. Programmtechnische Realisierung ausgewählter Teilaspekte

Persistenzhaltung:

Die Routinen für das Laden/Speichern von Stammdaten und Plandaten wurden mit Hilfe von Daten-(De)Sequentialisierungen in streams umgesetzt.
Da ein Übergang vom Plan- zum Stammdaten-Modus möglich sein soll, muß jede Plandaten-Datei zwangsläufig auch alle Stammdaten enthalten. Aus der Menge der Stammdaten läßt sich die Variable `CPlan::Alle` rekonstruieren. Die Sequenz, aus der beim Laden einer Plan-Datei eine Raum-Listen erzeugt wird, besteht aus einer Folge von Zahlen, in der die -1 eine Leerstunde bedeutet und eine Zahl größer gleich null die Position in `CPlan::Alle` bestimmt. Der so definierte Zeiger auf ein `CStunde`-Objekt wird in die Position der Raum-Liste kopiert, die durch die Nummer in der Sequenz eindeutig festgelegt ist. Eine weiterer

Typ einer Daten-Sequenz, ebenfalls durch die Position in `CPlan::Alle` identifiziert, beschreibt Änderungen der Nebenbedingungen einzelner Stunden in der Planungs-Phase wie z.B. die nachträgliche Beschränkung auf einen Tag oder auf einen bestimmten Raum. Wird für eine Stunde in `CPlan::Alle` eine solche Sequenz gefunden, werden die Eigenschaften dieser Stunde mit den Informationen der Sequenz überschrieben.

Manuelle Optimierung:

Für die manuelle Optimierung ist das **Schnappschuß**-Fenster verantwortlich. Es kann jederzeit aufgerufen werden, wenn kein Optimierungs-Verfahren aktiv ist.
Das Formular enthält ein Objekt vom VCL-Typ `TStringGrid`. Dies ist eine visualisierbare Tabelle von Strings. In der Ereignis-Behandlungs-Routine für das Ereignis `OnDrawCell` kann eine Zelle der Tabelle (bzw. des Gitters) nach individuellen Wünschen neu gezeichnet werden. In Bezug auf einen Lehrer, eine Klasse oder einen Raum wird dabei eine Zuordnung zwischen der Position der Zelle im Gitter und dem entsprechenden time-slot des Plan-Objektes vorgenommen.
Nach Markierung einer Zelle durch Maus oder Tastatur können über ein Popup-Menü prescheduling-Eigenschaften für die Stunde gesetzt oder eine Reihe von Funktionen ausgeführt werden.

Ausgabe:

Im aktuellen Stand der Software ist die Ausgabe der Plan-Daten an Winword implementiert. Über einen Aufruf des OLE-Automations-Servers vor Word, wird ein VBA-Zugriff von XXX auf die Funktionen dieser Textverarbeitung ermöglicht.

Hilfe-System:

Die Hilfe-Texte liegen in Form von HTML-Dokumenten vor.
Mit Hilfe des `ShellExecute`-Befehls kann der im System registrierte Standard-Browser mit einem HTML-Dokument als Parameter aufgerufen werden:

```
void __fastcall TForm1::Hilfe(String filename)
{
     void *i = ShellExecute(Handle,"open",(HelpPath + "\\" +
                    filename).c_str(),"","",SW_MAXIMIZE);
}
```

In der Funktion wird der übergebene Dateiname mit dem Hilfe-Pfad zu den Dokumenten verknüpft. Um die Kontext-Sensitivität der Hilfe-Aufrufe zu gewährleisten, wird der Member-Variablen `Hint` der visuellen Komponente, für die die Hilfe angefordert wird, der Name der HTML-Datei zugewiesen. Der Aufruf der Hilfe-Funktion erfolgt durch:

```
Hilfe(((TControl*)(HilfePM->PopupComponent))->Hint);
```

wobei `HilfePM->PopupComponent` eine hier nicht näher erläuterte globale Variable ist, die mit dem Zeiger auf das Objekt belegt wird, das den Hilfe-Aufruf abgesetzt hat[19].

[19] Bisher wurden einige Fenster der Stammdaten-Phase mit Hilfe-Texten versehen.

Vertretungs-Planung:

Die Vertretungs-Planung gestaltet sich relativ einfach, da ein Gebrauch von den vorhandenen Plan-Strukturen gemacht werden kann. Unter Verwendung der in 5.3 erläuterten Iterations-Varianten werden Verfügbarkeiten und Überschneidungs-Beziehungen ermittelt. Um die tägliche Erstellung der Vertretungs-Pläne zu beschleunigen, werden die fehlenden Lehrer in die Stammdaten-Datei mit aufgenommen.

4.4. Lösungsansatz: Stundenplan-Erstellung als Evolutions- Programm

4.4.1. Evolutionäre Algorithmen - ein Überblick

Evolution wird als kumulativer hochgradig paralleler Prozeß angesehen. Theorien, die hinreichend die Effizienz von Prozessen dieser Art erklären, sind jedoch noch nicht gefunden. Evolutionäre Algorithmen sind Verfahren, die bestimmte Aspekte der Natur herausgreifen und imitieren. Sie wurden etwa zeitgleich und unabhängig voneinander mit verschiedenen Schwerpunkten in Europa und den USA entwickelt. Allen gemeinsam ist eine Basis an naturanalogen Funktionsprinzipien. Dies sind die Strategien der natürlichen Evolution. Man faßt sie unter dem Sammelbegriff „Evolutionäre Algorithmen“ (EA) zusammen bzw. bezeichnet die resultierenden Programme als „evolutionäre“ oder „Evolutions - Programme“ (EP). [Mic]
Die Anwendung evolutionärer Algorithmen auf Optimierungs - Probleme liefert gute Ergebnisse. Dabei zeigt sich, daß EA's auf eine große Zahl verschiedenster Problemklassen anwendbar sind. Insbesondere werden Erfolge bei der Anwendung auf solche Probleme erzielt, die mit klassischen Verfahren nicht lösbar sind.

Die universelle Anwendbarkeit der EA's hat zur Folge, daß sich die Begriffe EA bzw. EP nur unscharf definieren lassen. Speziell die Tatsache, daß zunehmend auch diskrete Probleme mit EP's bearbeitet werden, erschwert die Begriffsdefinition. Daher soll zuerst der Versuch unternommen werden, die Funktionsweise von EA's in allgemeinster Form zu beschreiben, bevor in den Punkten 4.4.1.1 bis 4.4.1.3 auf spezielle Ausprägungen eingegangen wird.
Ziel dieses Abschnittes ist es, eine Grundlage zu schaffen für die Charakterisierung der in die Stundenplan-Software integrierten Optimierungs-Verfahren. Für dieses Algorithmen sollen in Abschnitt 4.4.2, ausgehend von bekannten Vor - und Nachteilen der etablierten Verfahren, Überlegungen bzgl. Anwendbarkeit und Effizienz angestellt werden.

Evolutionäre Algorithmen

Eine Menge von INDIVIDUEN - die POPULATION - lernt kollektiv durch die Prinzipien VERERBUNG, MUTATION und SELEKTION.
Ein Individuum ist eine Datenstruktur, die eine *potentielle* Lösung eines zu bearbeitenden Problems darstellt. Die Datenstruktur besteht aus Blöcken von Informationen - vergleichbar mit einem C++-struct oder einem Pascal-Record, in welchem jedem Informationsblock eine Variable zugeordnet ist. Jede Block - Information hat eine ganz bestimmte Bedeutung - analog der Tatsache, daß jedes Gen der DNS die Ausprägung eines bestimmten Teils unseres Organismus definiert - z.B. die Augenfarbe. Darüber hinaus hat jeder Block einen bestimmten Datentyp. Tauschen zwei Individuen untereinander Blöcke gleichen Typs aus (REKOMBINATION, VERERBUNG), können neue Individuen mit veränderten Eigenschaften

entstehen. Neue Individuen können auch durch zufälliges Verändern der Block-Informationen erzeugt werden (MUTATION). Die Individuen werden auch als Chromosomen bezeichnet.

Da die Anzahl der Individuen (POPULATIONSGRÖßE) i. Allg. begrenzt ist, erfolgt eine SELEKTION. Dabei haben die Individuen die größten Überlebenschancen, die die besten Lösungen im Sinne der Problemstellung repräsentieren. Die so neu formierte Population ist die neue Eltern - Population.

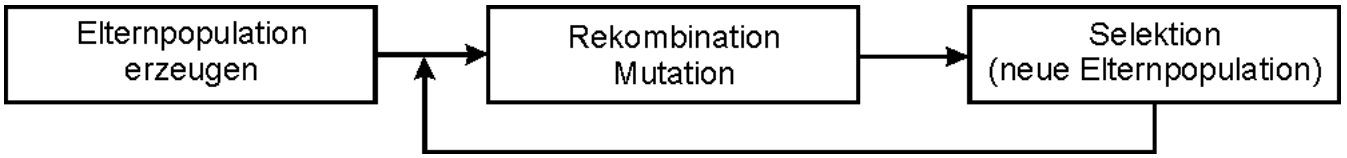


Abbildung 5: Einfaches Schema eines Evolutions - Algorithmus

In der Praxis der evolutionären Programmierung muß nun die jeweilige Problemstellung analysiert und in den Evolutions-Prozeß eingebettet werden. Eine fundamentale Rolle spielt dabei die sogenannte FITNEß - FUNKTION F. Die Fitneß - Funktion ist eine Abbildung aus dem Suchraum M in die Menge der reellen Zahlen.

$$F : M \rightarrow \Re$$

Dabei ergibt sich der Suchraum aus der Individuenstruktur und den Datentypen der einzelnen Block - Informationen. Jedes Individuum stellt nun einen bestimmten Zustand bzw. einen Punkt im Suchraum dar. Oftmals ist der Suchraum eingeschränkt durch problemspezifische Nebenbedingungen, denen die Individuen genügen müssen. Gesucht sind Individuen, bei denen F extremale Werte annimmt.
Häufig als Individuen anzutreffen sind s-dimensionale Parameter-Vektoren reeller Zahlen. In diesem Fall ist der Suchraum der $\Re^s$, die Fitneß - Funktion eine Abbildung $F : \Re^s \rightarrow \Re$ und ein Individuum ein Punkt im $\Re^s$.
Die Abbildungsvorschrift von F nimmt eine Bewertung der Güte eines Individuums vor. Aber natürlich stellt ein Individuum aufgrund eines guten Fitneß - Wertes nur dann eine gute Lösung dar, wenn die Fitneß - Funktion das tatsächliche Problem korrekt modelliert.
Es ergibt sich ein konkreteres Schema für EA's:

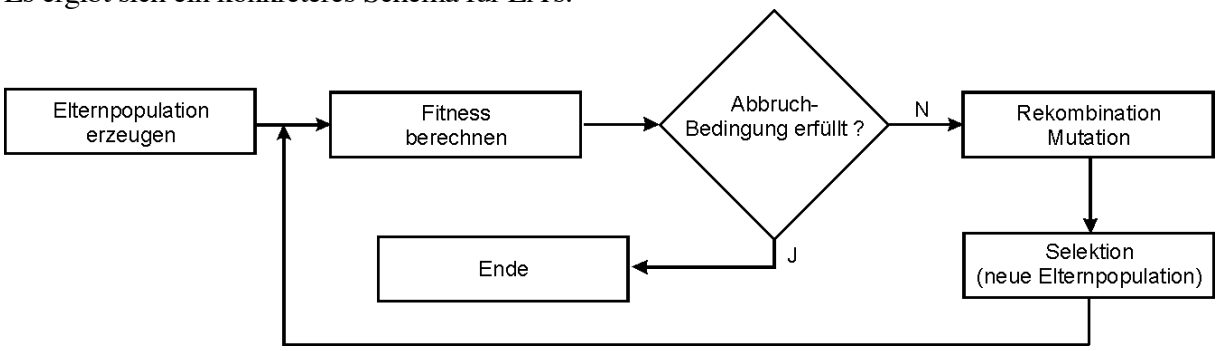


Abbildung 6: Erweitertes Schema eines Evolutions - Algorithmus

4.4.1.1. Mutations-Selektions-Verfahren

„Im Wesentlichen handelt es sich um die zufällige Veränderung von Systemparametern solange, bis eine Ziel - oder Kostenfunktion ein Minimum oder Maximum annimmt.“ [Kin,19]
Man wähle sich also ein Start - Individuum und verändere es durch Mutation - eine Abbildung $m : M \rightarrow M$. Wird ein besserer Fitneß - Wert erreicht, ersetzt das veränderte Individuum das Start - Individuum. Dieser Vorgang wird solange durchgeführt, bis eine Abbruchbedingung erfüllt ist.
Das Mutations-Selektions-Verfahren (MSV) ist also eine Spezialisierung des in Abbildung 6 skizzierten Schemas derart, daß die Population aus nur einem Individuum besteht und daher die Durchführung einer Rekombination nicht möglich ist.
Mutations-Selektions-Verfahren werden hauptsächlich in der numerischen Mathematik (Gleichungssysteme, Differentialgleichungen) und für Optimierungsprobleme im Bereich Organisation und Planung benutzt (Lagerhaltung, Stundenplanproblem, Handelsreisendenproblem).

Die klassische Veranschaulichung für dieses Verfahren ist ein Bergsteiger, der im Nebel versucht, den höchsten Gipfel einer Landschaft zu erklimmen. Dabei entspricht die Suche nach dem höchsten Berg der Suche nach dem Maximum der Fitneß - Funktion. Da der Bergsteiger stets nur bergauf wandert und nur soweit sieht, wie er seine Schritte setzen kann, läuft er Gefahr, lediglich auf dem Gipfel des nächst besten Hügels anzukommen und nicht auf dem höchsten Berg der Landschaft. Die Tatsache, daß er sich stets bergauf bewegt, ist also von entscheidender Bedeutung. Entspricht die Höhe dem Fitneß - Wert, findet sich ein Ausweg aus diesem Dilemma, wenn in begrenztem Umfang Fitneß - Verschlechterungen zugelassen werden:

„Dabei bedient man sich im wesentlichen zweier Grundschemata:
- Eine Verschlechterung der Fitneß ist mit einer gewissen - allerdings sehr kleinen - Wahrscheinlichkeit möglich (SIMULATED ANNEALING).
- Eine Verschlechterung der Fitneß ist stets möglich, aber höchstens bis zu einem maximalen Betrag (THRESHOLD ACCEPTING).“ [Kin, 46]

Neben dem Threshold Accepting stellt die SINTFLUT - METHODE eine ähnliche Ausprägung des zweiten Punktes dar.

4.4.1.2. Genetische Algorithmen

Genetische Algorithmen (GA) orientieren sich eng an den Gesetzen von Genetik und natürlicher Evolution. GA's entsprechen weitgehend dem Schema in Abbildung 6. Ihr charakteristisches Merkmal ist das Vorhandensein einer Population, deren Individuen in einem kollektiven Lernprozeß Informationen austauschen. Dieser Austausch (Rekombination, Vererbung, Kreuzung) wird, im Gegensatz zur Mutation, durch Transformationen höherer Ordnung vollzogen. Man spricht im allg. von Rekombinations- bzw. CROSSOVER- Operatoren (Kreuzung):

$$c_j : M \times ... \times M \rightarrow M$$

Motivation für diese Austauschvorgänge ist die Annahme, daß verschiedene aber ausgezeichnete Teile eines Individuums maßgeblich für gute Fitneß-Werte verantwortlich sein können. Wird nun durch die Verschmelzung zweier (oder mehrerer) Individuen zu einem

neuen eine Zusammenführung dieser „wertvolleren“ Bestandteile erreicht, kann dies einen großen Sprung bei der Suche nach dem Individuum mit dem optimalen Fitneß-Wert bedeuten. Als weiterer Vorteil der GA's gegenüber den Mutations-Selektions-Verfahren gilt die (hohe) Zahl der gleichzeitig den Suchraum durchstreifenden Individuen. Damit steigt die Wahrscheinlichkeit, daß eines dieser Individuen in einen Bereich des Suchraumes gelangt, der den Zustand mit dem optimalen Fitneß - Wert enthält. Allerdings schlägt sich das Halten einer Population in höherem Programmieraufwand und größerem Bedarf an Ressourcen nieder. Ferner bieten Populationen mit vielen Individuen vor allem erst dann einen Vorteil, wenn die Optimierungs-Programme parallelisiert werden können - im Idealfall steht für jedes Individuum ein separater Prozessor zur Verfügung.
Das biologische Vorbild der crossover - Operatoren ist das crossing over[20] homologer Chromosomen bei der Meiose. Der crossover - Operator hätte die Form $c_2 : M \times M \rightarrow M$.
Eine einfache Form eines crossover - Operators für GA's ist in der folgenden Abbildung dargestellt.

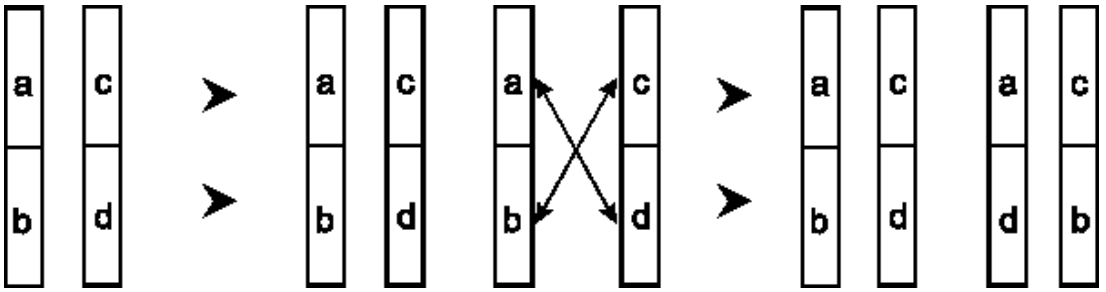


Abbildung 7: Einfaches Beispiel für einen crossover - Operator

Zwei Individuen werden kopiert. Die Kopien werden an einer bestimmten Stelle (cross point) - hier in der Mitte - geteilt und kreuzweise miteinander verbunden. Es entstehen zwei neue Individuen mit (wahrscheinlich) veränderten Eigenschaften, die nun mit den Individuen der Eltern - Population in Konkurrenz treten. Da die Populationsstärke i. allg. begrenzt ist, muß also eine Auslese bzw. Selektion vorgenommen werden. Dabei sollten die Individuen mit den besten Fitneß-Werten *die größten Chancen* haben, in die neue Eltern-Population aufgenommen zu werden. Die wortwörtliche Einbettung des vielzitierten „survival of the fittest“ in den GA führt oft zu schlechteren Ergebnissen. Individuen mit hohen Fitneß-Werten überleben also nur mit höherer Wahrscheinlichkeit die Selektion und verdrängen nicht rigoros alle „schlechteren“ Chromosomen. Dies impliziert natürlich auch die Möglichkeit, daß Individuen mit guter Fitneß aus der Population gelöscht werden. Eine solche Löschung bietet dem Algorithmus die Gelegenheit, Individuen von ungünstigen lokalen Optima abzuziehen und neue Bereiche des Lösungsraumes zu untersuchen.

4.4.1.3. Weitere evolutionäre Algorithmen

Weitere bekannte evolutionäre Algorithmen sind die GENETISCHE PROGRAMMIERUNG und die EVOLUTIONSSTRAGEGIEN. Auch diese Verfahren sind Spezialisierungen des in Abbildung 6 dargestellten Schemas.
Evolutionsstrategien werden zur Funktionsoptimierungen angewandt, wobei die Individuen reellwertige Parameter-Vektoren darstellen. Die Beschränkung der Eigenschaften der

[20] Das englische „crossing over“ ist ein biologischer Fachbegriff und läßt sich nur schlecht durch das umgangsprachliche „Kreuzen“ übersetzen. Bei der Kreuzung zweier Individuen ist ein crossing over lediglich mit einer bestimmten Wahrscheinlichkeit zu erwarten.

Individuen auf reelle Zahlen machen diese Verfahren für eine Bearbeitung des Stundenplanproblems ungeeignet.
Die genetische Programmierung unterscheidet sich in den Funktions-Prinzipien nur wenig von den GA's. Eine gesonderte Einordnung dieser Verfahren wird wegen dem speziellen Anwendungsgebiet und den sich daraus ergebenden Individuen-Strukturen und Operatoren vorgenommen. Genetische Programmierung versucht im wesentlichen eine automatische Erzeugung von Computerprogrammen und Berechnungsvorschriften zu realisieren.

4.4.2. Nachdenken über die Evolution, Vorbetrachtungen

Thema dieses Abschnittes sind die Fragen: Warum sollte erstens ein evolutionärer Algorithmus angewandt werden und zweitens, wenn ein EA, welche Art und warum.
Das SPP ist (wahrscheinlich) np-vollständig. Somit ist i. Allg. eine sichere Lösungsfindung in vertretbarer Zeit nicht möglich und es bieten sich naturanaloge Verfahren an. Allerdings sollten die klassischen Hill-Climbing-Verfahren nicht völlig außer acht gelassen werden, da sich bestimmte Fragestellungen in beiden Typen von Verfahren wiederfinden.

Ausgangspunkt bei der Wahl des Optimierungs-Verfahrens ist der Versuch, ein Modell für den Lösungsraum des SPP zu entwickeln. Anschließend soll aus den Eigenschaften dieses Modells die Eignung bzw. Unbrauchbarkeit bestimmter Verfahren abgeleitet werden.
Der grundlegende Unterschied zwischen klassischen Optimierungs-Problemen und der Optimierung eines Stundenplanes, besteht in dem diskreten Charakter des SPP und den nicht stetigen Ziel-Funktionen. Diesen Eigenschaften muß in der Entwicklung eines Lösungsraum-Modells Rechnung getragen werden. Die folgende Abbildung veranschaulicht den Lösungsraum in Form eines Graphen[21].

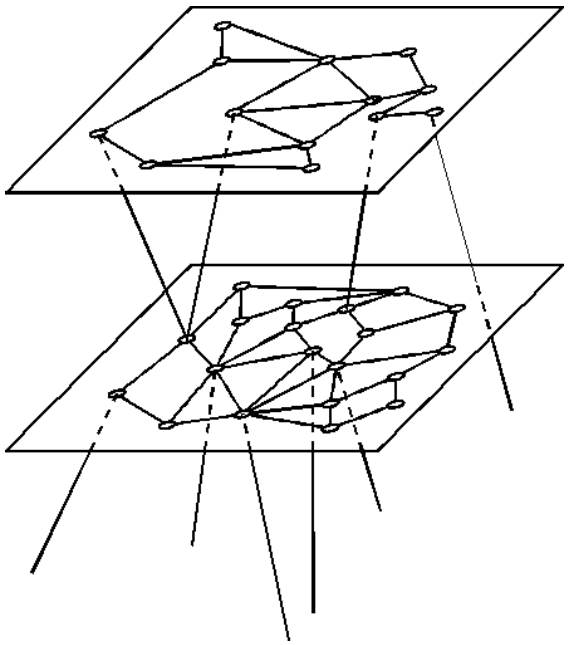


Abbildung 8: Modell des Lösungs-Raumes eines Stundenplan-Problems

Jeder Knoten stellt einen bestimmten Zustand des Planes dar und definiert sich durch die Zuordnung aller Stunden zu der Menge der time-slots. Der Austausch zweier Stunden erzeugt

[21] Diese Abbildung stellt eine starke Vereinfachung dar, demonstriert aber dennoch wesentliche Eigenschaften realer Plan-Strukturen.

einen neuen Zustand[22]. Ein solcher Zustands-Übergang wird im Graphen-Modell durch eine Kante symbolisiert. Es werden nur Zustände betrachtet, die einen gültigen Plan darstellen. Genau genommen durchläuft ein Plan in der Software kurzzeitig auch Zustände, die einen ungültigen Plan repräsentieren. Soll z.B. eine Doppelstunde verschoben werden, tritt zwangsläufig eine zeitweilige Trennung der beiden Stunden auf, da die Stunden nur *nacheinander* verschoben werden können. Eine noch höhere Zahl ungültiger Zustände durchläuft der Plan beim Verschieben von Kursstunden.
In jedem Zustand lassen sich die Werte der Ziel-Funktionen bestimmen, also die Anzahl der Leerstunden, das Maß für die Stundenverteilung u.s.w.. Für die obige Darstellung wird nur eine Ziel-Funktion angenommen - z.B. die Klassen-Leerstunden (KLS). Die beiden Ebenen enthalten jeweils alle die Zustände, für die sich der selbe Wert der KLS ergibt. Weiterhin repräsentiere die obere der Ebenen den besseren Wert. Ein Optimierungs-Verfahren versucht nun, durch die Verlagerung von Stunden zwischen den time-slots Ebenen mit besseren Ziel-Funktions-Werten zu erreichen. Dabei zeigt sich, daß sich mit fortschreitender Optimierung die Zahl der Zustands-Übergänge zu besseren Werten vermindert. Aus dieser Tatsache leitet sich folgende zentrale Frage ab: Wird die Zahl dieser Zustands-Übergänge gegen null konvergieren, obwohl noch kein Optimum der Ziel-Funktion erzielt wurde? Die analoge Fragestellung im Kontext eines klassischen Hill-Climbing-Verfahrens lautet: Konvergiert das Verfahren nur gegen den nächstgelegenen lokalen Extremwert oder besteht die Möglichkeit, das globale Optimum zu erreichen?
Die trivialer Weise nicht vorhandene Kenntnis des globalen Optimums und die Gefahr der Konvergenz gegen lokale Extrema erfordert bestimmte Strategien in der Optimierung. An diesem Punkt lohnt sich ein Blick in die Natur. In [Kin, S. 9] wird dazu treffend bemerkt: „Optimierungen sind ‚natürlich', was bedeutet, daß die Natur sich ihrer bedient. Dies gilt sowohl für die unbelebte Welt der Physik wie für die belebte Welt der Biologie." Wenn die Evolution das überaus komplexe Ökosystem der Erde hervorbringen kann, liegt die Vermutung nahe, daß Evolutions-Mechanismen auch erfolgreich auf vergleichsweise einfache Probleme angewandt werden können.
Worin besteht der Zusammenhang zwischen einem Ökosystem und dem Lösungsraum eines Stundenplanes? Ein Individuum lebt in einer Umwelt, welche durch ihre Bedingungen beschrieben werden kann. Ein gültiger Stundenplan ist ein Element des Lösungsraumes, der sich durch das zu Grunde liegenden SPP und dessen Bedingungen definiert. Die Strukturen von Lösumgsräumen und Ökosystemen lassen sich nicht mit Hilfe stetiger Funktionen beschreiben. Die bei klassischen Problemen gängige Veranschaulichung des Lösungsraumes durch eine harmonische Hügel-Landschaft ist hier nicht anwendbar. Vielmehr entziehen sich die Strukturen von Ökosystemen aufgrund ihrer Komplexität noch immer weitgehend unserer Kenntnis. Ähnliches gilt für die Beschaffenheit des Lösungsraumes eines SPP. Die Anpassung eines Lebewesens an seine Umwelt, also die „Bewegung" in dem Lösungsraum Ökosystem, basiert auf den Mechanismen der Evolution. Dabei ist es vor allem die Mutation, die den Evolutions-Prozeß am Leben erhält: „Evolution ist also der spezifische Filterungsprozeß des durch die Vielzahl von Mutationen bereitstehenden Spektrums der Möglichkeiten - ein Filterungsprozeß im Hinblick auf eine bestimmte vorteilhafte Eigenschaft oder Fähigkeit, die das Überleben in der betreffenden Überlebensnische verbessert." [Cra, S. 71] Die Charakteristika dieses Filterungsprozesses gilt es genauer zu untersuchen.
„Der Kampf ums Überleben" ist eine folgenschwere Fehlübersetzung[23] von Darwins „struggle for life" und führte letztlich zu Fehlinterpretationen des vielzitierten "survival of the fittest". [Ste] Das *Überleben der Stärksten* ist kein Dogma, sondern eine von

---

[22] Die Verschiebung *einer* Stunde in einen leeren time-slot stellt sich in der Implementaton als Austausch von *zwei* `CStunde`-Objekten dar - ein Austausch von einer leeren und einer nicht-leeren Stunde.
[23] „struggle for life" bezeichnet eher die tägliche Mühsal und Schinderei - die Auseinandersetzung mit den täglichen Unbilden der Existenz.

Wahrscheinlichkeiten bestimmte Entwicklungs-Tendenz. Je besser Individuen aufgrund ihrer Eigenschaften an die Umwelt angepaßt sind, desto größere Überlebens-Chancen haben sie. Trotzdem besteht auch für weniger angepaßte Individuen eine Überlebens-Wahrscheinlichkeit. Eine breites Spektrum von Individuen, die sich durch ihren Anpassungsgrad unterscheiden, kann im Fall einer drastischen Änderung der äußeren Lebensbedingungen das Überleben der gesamten Art sichern, wenn sich ein vorher schlechter angepaßtes Individuum für den neuen Lebensraum als geeigneter erweist. Desweiteren kann sich ein solches Individuum als Ausgangspunkt der Entstehung verschiedener Unterarten herausstellen, die besser an den Lebensraum angepaßt sind oder sich eine völlig neue ökologische Nische erschließen. Durch die Existenz von Individuen unterschiedlicher Eigenschaften kann die Evolution also als Verfahren mit Back-Tracking-Charakteristiken verstanden werden. Der Informations-Speicher für das Back-Tracking ist die Summe der Erbinformationen der Individuen einer Population.
Die Entwicklung mehrerer von einer Art abstammenden Unterarten findet eine Analogie in der Vermeidung von asymptotischem Optimierungsverhalten in Gegenden des Suchraumes, welche keine globalen Extrem-Werte bzgl. einer Fitneß-Funktion aufweisen. Wird es einem Stundenplan ermöglicht, in frühere Zustände zurückzukehren, kann durch eine veränderte Folge von Zustands-Übergängen ein neuer Bereich des Lösungsraumes nach Extrem-Werten abgesucht werden.
Während bei genetischen Algorithmen die Existenz einer Population Back-Tracking-Charakteristiken impliziert, sind beim Mutations-Selektions-Verfahren ergänzende Strategien notwendig, um die unerwünschte Konvergenz gegen lokale Extremwerte zu vermeiden. Die Wahl des Optimierungs-Verfahrens (GA oder MSV) hängt ab von Effizienz und Anwendbarkeit des Verfahrens selbst und vom Aufwand bei der Programmierung:

	GA	MSV
Wahrscheinlichkeit, globales Optimum zu erreichen	hoch	gering
Population	J	N
Operatoren	mutation, crossover	mutation
Programmieraufwand	hoch	gering

Tabelle 6: GA vs. MSV

Der Vergleich zwischen GA und MSV scheint, abgesehen vom Programmier-Aufwand, für den GA zu sprechen. Die Charakteristiken des SPP relativieren jedoch die Aussagen obiger Tabelle. Durch die Ergänzung des MSV mit Strategien wie Simulated Annealing, Threshold Accepting oder Sintflut-Verfahren steigt die Wahrscheinlichkeit, das globale Optimum zu erreichen. Entscheidend bei der Wahl des Verfahrens ist jedoch die Tatsache, daß die Mutation den eigentlichen Impuls der Evolution darstellt und darüber hinaus ein crossover-Operator zwischen zwei Stundenplänen, wenn überhaupt, nur mit begrenztem Erfolg einsetzbar ist. Das Problem besteht darin, daß sich zwischen zwei Plänen nicht ohne weiteres Teilstrukturen austauschen lassen, wie etwa zwischen zwei Bit-Ketten. Ein mögliches Verfahren ist in [Erb] zu finden und soll hier kurz dargestellt werden.

Die Stundenpläne zweier Klassen werden als Individuen betrachtet[24]. Die n Stunden der Klasse seien durch Zahlen von 1 bis n identifiziert und können in ihrer zeitlichen Abfolge als Permutationen betrachtet werden.

[24] Für Räume und Lehrer ist eine analoge Herangehensweise möglich.

Für ein stark vereinfachtes Beispiel möge $n = 8$ gelten. Leerstunden werden nicht betrachtet und die Anzahl der time-slots wird ebenfalls auf *8* festgelegt. Es werden weiterhin nur die Verfügbarkeiten der Klasse betrachtet.

Plan 1:	*5*	*4*	*8*	*2*	*3*	*1*	*6*	*7*
Plan 2:	*1*	*3*	*7*	*6*	*5*	*4*	*8*	*2*

Existiert für die Permutation eine Zyklen-Darstellung, kann jeweils die Teilmenge der Stunden, die in dem Zyklus enthalten ist, zwischen den Plänen verschoben werden. Da die Menge der am Zyklus beteiligten Stunden in beiden Plänen identisch ist, ist die Verfügbarkeit für die Klasse gesichert.
Die obige Permutation läßt sich als Produkt elementfremder Zyklen schreiben:

$$\begin{pmatrix} 1435 \\ 4351 \end{pmatrix} \begin{pmatrix} 2687 \\ 6872 \end{pmatrix}$$

Damit ergibt sich eine Reihe von Möglichkeiten, neue Plänen zu erzeugen. Ein Beispiel:

Plan 3:	*1*	*3*	*8*	*2*	*5*	*4*	*6*	*7*

Jede weitere Permutation der Stunden 1, 3, 5, 4 ergibt einen weiteren Plan. Sollen zusätzlich die Verfügbarkeiten der Lehrer berücksichtigt werden, muß in jedem Plan der beteiligten Lehrer ein Zyklus der selben Menge von time-slots vorhanden sein. Gleiches gilt für die beteiligten Räume. Da stets alle Verfügbarkeiten Beachtung finden müssen, ist die Wahrscheinlichkeit, einen Zyklus zu finden, gering. So wird auch in [Erb] konstatiert, daß, wenn überhaupt Zyklen gefunden wurden, diese nur sehr kurz waren. Es ist klar, daß sich kurze Zyklen durch einige Verschiebungen von Stunden ersetzen lassen. Diese Aussage wird durch einen Satz der Algebra gestützt, nach welchem sich jede Permutation als Produkt nicht notwendig elementfremder Transpositionen schreiben läßt. [Lug, S. 87] Allerdings sinkt die Wahrscheinlichkeit, daß ein Zyklus auch tatsächlich durch Transpositionen ersetzt wird, mit dessen Länge.

Der Mangel an praktikablen crossover-Operatoren und die Tatsache, daß eine sinnvolle parallele Verarbeitung von Individuen einer Population nur durch hohe Hardware-Voraussetzungen realisiert werden kann, führt schließlich zu der Entscheidung, von einer Verwendung genetischer Algorithmen als Optimierungs-Verfahren abzusehen. Statt dessen erfolgt die Optimierung mittels MSV und den Optimierungs-Strategien Tabu Search, Simulated Annealing und dem Sintflut-Verfahren.

5. Optimierung

5.1. Programm-Ablauf und allgemeine Grundregeln der Optimierung

Ausgangspunkt jeder Optimierungs-Phase ist ein gültiger Plan. Bei Erhalt der Gültigkeit des Planes werden durch heuristische Mischverfahren Zustands-Übergänge vorgenommen. Verschiedene Optimierungs-Strategien steuern die Auswahl der Übergänge und versuchen auf dieses Weise eine Optimierung der Ziele zu erreichen.
Die folgende Grafik stellt den globalen Optimierungsprozeß und die Einordnung der Teil-Optimierungen in diesen dar.

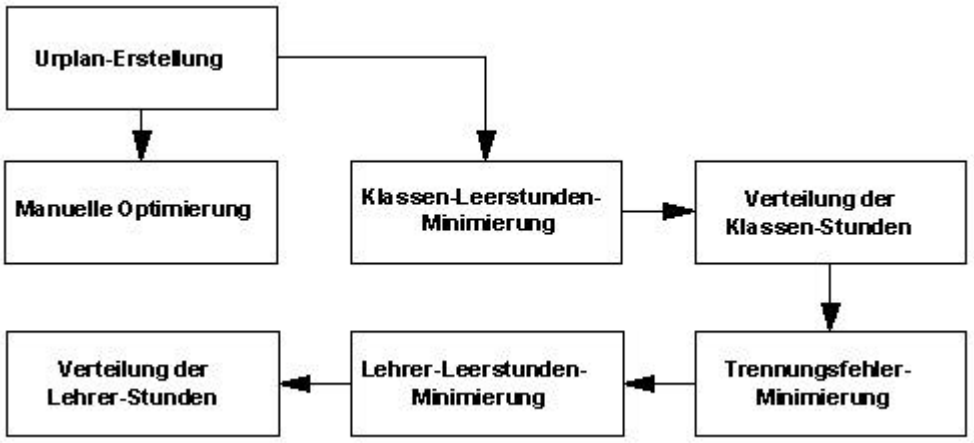


Abbildung 9: Ablauf der Optimierungs-Phasen

Die Urplan-Erstellung kann als erste Phase der Optimierung betrachtet werden, da hier bereits einige Nebenbedingungen für die Erzeugung eines gültigen Planes beachtet werden müssen. Wie aus der Abbildung zu entnehmen ist, sind zwei Vorgehensweisen für die Optimierung der weiteren Teil-Ziele möglich - eine automatische und eine manuelle Variante. Die automatische Optimierung folgt einer fest vorgegebenen Reihenfolge. Dies bedeutet, daß der in den vorangegangenen Phasen jeweils erarbeitete Optimierungs-Zustand stets erhalten bleibt. Eine Optimierung der Trennungsfehler (TF) verschlechtert also nicht die Ergebnisse von Klassen-Stunden-Verteilung (KSV)[25] und Klassen-Leerstunden-Minimierung (KLS-Minimierung), ignoriert jedoch alle folgenden Ziele. Während verschiedener Test-Läufe hat sich gezeigt, daß die schrittweise Optimierung einzelner Ziele effizienter ist. Je mehr Nebenbedingungen ein Verfahren gleichzeitig berücksichtigen muß, desto seltener sind Zustands-Übergänge, die die Werte der Ziel-Funktionen verbessern. Die Rangfolge kann also als Empfehlung für den Nutzer verstanden werden, läßt sich jedoch problemlos durch die manuelle Optimierung ersetzen.

Erfahrungen mit EA's zeigen, daß sich die Effizienz derartiger Algorithmen in dem Maße erhöht, wie sich die benutzten Datenstrukturen und deren Operatoren durch eine stärkere Spezialisierung auf das jeweilige Problem hin auszeichnen. Die Möglichkeit, EA's auf einen großen Bereich verschiedenartiger Problemstellungen anwenden zu können, geht hingegen zumeist mit einer Verschlechterung der Effizienz einher. [Mic, S. 289 ff]
Welchen Einfluß haben spezialisierte Datenstrukturen und Operatoren auf die Optimierung? Wie eine Nebenbedingung schränkt ein spezialisiertes Verfahren den Lösungsraum ein. Ein

[25] Das entsprechend Optimierungs-Ziel für die Lehrer wird mit LSV abgekürzt

Effizienz-Gewinn wird jedoch nur dann erzielt, wenn die durch die Spezialisierung entfernten Plan-Zustände keine Bedeutung für die Optimierung haben. Im folgenden werden zwei Beispiele für solche Spezialisierung dargestellt:

- *Vermeidung des Austausches identischer Stunden:*
  Es wird zwar tatsächlich eine Verschiebung von Stunden zwischen time-slots vorgenommen, allerdings hätten diese Übergänge keine Wirkung auf die Werte der Ziel-Funktionen.
- *Nutzung der Variable* `CStunde::RaumL`
  Da die Menge der Räume, in welche eine Stunde gesetzt werden kann, prinzipiell im voraus definiert wird, werden automatisch viele erfolglose Setz-Tests vermieden. Die Variable `CStunde::RaumL` erspart also der zentralen Funktion `CPlan::SetzTest` die Überprüfung, ob eine bestimmte Stunde in einen bestimmten Raum gesetzt werden darf.

Neben der Einschränkung des Suchraums durch spezialisierte Algorithmen und Datenstrukturen macht sich jede Form von Optimierung bei der Effizienz des globalen Optimierungs-Prozesses bemerkbar. Dies können z.B. effiziente Sub-Algorithmen oder Details in der Implementierung sein. Von großer Bedeutung beim Übergang in die Planungs-Phase ist z.B. die ausschließliche Verwendung von Integer-Variablen. Eine weitere Optimierung ergibt sich aus der Tatsache, daß der Austausch von Stunden stets nur einen Bruchteil des gesamten Planes verändert. Die Aktualisierung der Ziel-Funktions-Werte erfolgt also nur für die Lehrer und Klassen, deren Optimierungs-Zustand sich tatsächlich geändert hat.
In vielen Situationen erfolgt die zufällige Auswahl eines Elementes aus einer Menge. Diese Elemente können Stunden, Räume oder time-slots sein. Als Beispiel denke man sich eine Stunde, für die ein freier Raum benötigt wird. Damit bereits belegte Räume nicht mehrfach per Zufallswahl überprüft werden, wird die betreffende Raum-Menge vorher in eine Liste kopiert, aus der bereits getesteten Elemente entfernt werden können.
Generell sollte die allgemeine Strategie lauten, *jede* Möglichkeit der Optimierung auszunutzen.

5.2. Urplan - Erstellung

Alle Unterrichtsstunden werden zufallsbasiert in die vorhandenen Räume plaziert. Die einzige Bedingung, die es einzuhalten gilt, ist die Erzeugung eines *gültigen* Planes. Dies wiederum schließt die Beachtung einer Reihe von Nebenbedingungen ein, die durch den Benutzer in der Stammdaten-Phase definiert wurden.

Für die Urplan-Erstellung wurde eine Rangfolge von verschiedenen Typen von Stunden ermittelt. Dieses Rangfolge leitet sich aus dem Schwierigkeitsgrad ab, für die jeweilige Stunde einen möglichen Platz in dem Plan zu finden. Z.B. wird für Mehrfachstunden ein zusammenhängender Bereich von time-slots benötigt. Für Kursstunden hingegen müssen in verschiedenen Raum-Listen zu parallelen Zeiten freie time-slots zur Verfügung stehen. Die aufgeführte Rangfolge spiegelt die Erfahrungen wieder, die während Tests der Urplan-Erstellung gemacht wurden.

Die Rangfolge der Stunden-Typen bei der Urplan-Erstellung lautet:

1. Stunden, die fest in Raum und Zeit sind
2. zeitfeste Stunden, die in höchstens 2 Räumen stattfinden können
3. zeitfeste Stunden, die in höchstens 4 Räumen stattfinden können
4. restliche zeitfeste Stunden
5. Mehrfachstunden deren Block-Stunden-Anzahl mindestens 3 beträgt (keine Kursstunden)
6. einfache und mehrfache Kursstunden
7. Doppelstunden, die in höchstens 5 Räumen stattfinden können
8. restliche Doppelstunden
9. restliche Einfachstunden

Für die zu plazierenden Stunden werden per Zufall Zeiten ermittelt. Anschließend wird einer der für diese Stunde vorgesehenen Räume ausgewählt - ebenfalls per Zufall. Die Funktion `CPlan::SetzTest` überprüft, ob das Setzen der Stunde zu zeitlichen Überschneidungen führt oder sonstige Nebenbedingungen verletzt. Bei Kurs- und Mehrfachstunden müssen stets alle Stunden der Stunden-Definition gesetzt werden können. Im Erfolgs-Fall wird die Stunde in die entsprechende Raum-Liste plaziert. Tritt ein Problem auf, wird die Suche nach verfügbaren time-slots und Räumen fortgesetzt. Wird eine bestimmte Anzahl von Versuchen überschritten, gilt der Versuch einen Stundenplan zu erstellen, als gescheitert. In diesem Fall werden alle bis dato gesetzten Stunden mittels der Funktion `CPlan::ReInit` wieder aus den Raum-Listen entfernt und ein neuer Versuch, einen Urplan zu erstellen, wird vorgenommen. Dieser Algorithmus ist in der Methode `CPlan::InitUrPlan` implementiert. Sie enthält als Ergänzung ein Verfahren, welches versucht, eine Belegung der ersten Stunden aller Klassen an allen Tagen zu ermitteln. Der Programm-Nutzer hat dabei die Möglichkeit zu entscheiden, ob eine Urplan-Erstellung auch dann als erfolgreich angesehen wird, wenn noch nicht alle erste Stunden belegt sind. Liegt ein gültiger Urplan vor, können die weiteren Optimierungs-Ziele bearbeitet werden.

5.3. Optimierung mittels Mischverfahren

Heuristische Misch-Verfahren sind die Grundlage aller hier dargestellten Optimierungs-Algorithmen. In Bezug auf das Modell des Lösungsraumes in Abschnitt 4.4.2 sorgen diese Verfahren bei Erhalt der Gültigkeit des Planes für Zustands-Übergänge. Ihre Aufgabe ist es, schrittweise kleine Veränderungen des Stundenplanes vorzunehmen.
Ein weiterer bedeutsamer Effekt dieser Übergänge kann in der Umsortierung des Planes bestehen, welche eine Änderung von Überschneidungsbeziehungen verursacht und somit weitere Möglichkeiten der Optimierung bietet. Die Durchmischung eines Planes kann als Analogon zur Mutation im Evolutions-Prozeß verstanden werden. Während in der Natur die Selektions-Mechanismen für die Gerichtetheit der Entwicklung der Arten verantwortlich sind, übernehmen bei der Plan-Optimierung bestimmte Optimierungs-Strategien diese Rolle (siehe Abschnitte 4.4.1.1 und 5.5).

In diesem Abschnitt werden die Verfahren als solche vorgestellt. Der Umfang der Mischverfahren in der Implementation beträgt insgesamt rund 3000 Zeilen Programmcode. Aus diesem Grund kann hier keine erschöpfende Erläuterung der Algorithmen vorgenommen werden.
Die Algorithmen basieren häufig auf einer Iteration über die time-slots des vom Benutzer gewählten Zeit-Fensters. Dabei werden die entsprechenden Stunden-Container der in `CPlan::RL` gespeicherten Raum-Listen auf Stunden mit bestimmten Eigenschaften

überprüft. Es lassen sich zwei Iterations-„Richtungen“ unterscheiden: Iterationen über Zeiten und über Räume. Wird über eine Menge von Räumen iteriert, steht die Position eines time-slots fest und die Räume werden zu dieser Zeit auf Stunden mit bestimmten Attributen untersucht. Soll z.B. festgestellt werden, ob eine Klasse zu einer bestimmten Zeit Unterricht hat, bietet sich dieser Iterations-Typ an. Bei einer Iteration über die Zeit wird in einem festen Raum über die time-slots des Zeit-Fensters iteriert. Diese Iterations-Art währe etwa bei der Suche nach einem freien time-slot eines bestimmten Raumes hilfreich.
Ein weiteres wichtiges Element in den Algorithmen ist die zufällige Positionierung von Stunden bzw. die zufalls-basierte Suche nach Stunden mit bestimmten Attributen. Für diesen Zweck wurde eine einfache Kapselung des Zufallszahlen-Generators in einer globalen Hilfsfunktion vorgenommen:

Bezeichner	`Ra`
Parameter	`int von` `int bis`
Rückgabe-Typ	`int`
Kurz-Beschreibung	– liefert eine Zufallszahl im Intervall [`von`, `bis`]

Als kombiniertes Beispiel aus Raum-Iteration und Zufallselementen möge der folgende konstruierte Quelltext-Abschnitt dienen. Als Bezugsrahmen wird die Klasse `CPlan` angenommen. Damit sind die Variablen `Stunden`, `Tage`, `R` und `RL` entsprechend definiert.

```
...
0     TList *pRaum;
1     int stunde, tag;
2     CStunde *pS;
3     stunde = Ra(1,Stunden);       //zufällige Ermittlung einer Stunde
4     tag = Ra(1,Tage - 1);         //zufällige Ermittlung eines Tages
5     for (int r = 0; r < R; r++)   //Iteration über alle Räume
6     {
7           pRaum = (TList*)(RL->Items[r]);                //die Raum-Liste
8           pS = (CStunde*)(pRaum->Items[tag*13 + stunde]);//die Stunde
9           //falls die Stunde nicht leer und nicht zeitfest ist:
10          if ((pS->Lehrer != -1) && (pS->Fest != ZEIT)
                  && (pS->Fest != RAUMZEIT))
11          {
                  ...
            }
...
```

In den Zeilen 3 und 4 wird per Zufall ein time-slot ausgewählt. Die Tag-Stunde-Darstellung dieser Zeit ermöglicht im Gegensatz zu der Listen-Darstellung das Ausblenden der nicht durch das Zeit-Fenster erfaßten time-slots. Zeile 10 selektiert nur die Stunden der gegebenen Zeit, die nicht leer sind und die nicht durch ein prescheduling auf eine bestimmte Zeit festgelegt sind. Die Erfüllung der letzteren Bedingung währe z.B. Voraussetzung, wenn eine Stunde für eine Verschiebung ausgewählt werden müßte.

Es wurden drei Mischverfahren implentiert - jeweils repräsentiert durch eine Methode der Klasse `CPlan`:

a) `CPlan::Austausch`
b) `CPlan::Tausch`
c) `CPlan::KLSMinimierung`

Bezeichner	`CPlan::Austausch`

Parameter	`int typ`
	`int klassenLimit`
	`int tausch`

Rückgabe-Typ	`void`
Kurz-Beschreibung	– Austausch von Stunden einer Klasse; höchstens `tausch`-mal – Verwendung als Misch-Verfahren für die Ziele KLS und KSV und für die direkte Optimierung der Ziele TF, LLS und LSV – austauschbar sind: a) Einzel-Fachstunden und Einzel-Fachstunden b) Einzel-Fachstunden und Einzel-Kursstunden c) Einzel-Fachstunden und Mehrfachstunden – `typ` bestimmt das Optimierungs-Ziel und die Optimierungs-Strategie, welche entscheiden, ob der jeweilige Austausch akzeptiert wird

Der Austausch von Stunden erzwingt komplexe Operationen in der Menge der Raum-Listen. Eine Fülle von Konstellationen, die während des Austausches auftreten können, muß beachtet werden. An einem Beispiel soll der Austausch von Einzel-Fachstunden und Einzel-Kursstunden demonstriert werden (siehe auch folgende Abbildung):
Für eine Klasse wurde eine Einzel-Fachstunde *s1* zur Zeit *t1* und eine Einzel-Kursstunde *s2* zur Zeit *t2* gefunden. In den Kurs sind zwei weitere Klassen involviert. Die Fachstunden werden im folgenden mit *s1a*, *s1b*, *s1c* und die Kursstunden mit *s2a*, *s2b* und *s2c* bezeichnet. Das Problem besteht nun darin, die drei zeitparallelen Einzel-Fachstunden der Zeit *t1*, an die Zeit *t2* zu plazieren und die drei zeitparallelen Kursstunden an die Zeit *t1*. Unter der Bedingung des Erhaltes der Gültigkeit des Planes müssen folgende Voraussetzungen gelten:

1. Die drei Klassen müssen zur Zeit *t1* tatsächlich nur Einzel-Fachstunden haben.
2. Jeder Lehrer der Fachstunden muß zur Zeit *t2* verfügbar sein.
3. Jeder Lehrer der Kursstunden muß zur Zeit *t1* verfügbar sein.
4. Für die Fachstunden muß zur Zeit *t2* und für Kursstunden zur Zeit *t1* je eine Menge von drei Räumen zur Verfügung stehen, in welche die Stunden plaziert werden können.
5. Das zeitliche prescheduling aller Stunden muß beachtet werden.

Der vierte Punkt weist auf die Möglichkeit hin, daß eine Stunde der Zeit *t2* in einen Raum plaziert werden könnte, der von einer der Fachstunden der Zeit *t1* belegt ist. Bei der Überprüfung der möglichen Räume für die Kursstunden dürfen sich die Fachstunden also nicht mehr in den Raum-Listen befinden, da sonst die Funktion `CPlan::SetzTest` signalisiert, daß die Kursstunde in diesen Raum nicht plazierbar ist. Umgekehrt kann für die Fachstunden der Setz-Test für (potentiell ehemalige) Räume der Kursstunden scheitern. Aus diesem Grund müssen alle zu verschiebenden Stunden aus den Raum-Listen entfernt und durch Leerstunden ersetzt werden. Für diese Ersetzung wird die Variable `CPlan::LeerStunde` benutzt. Diese Ersetzung erfordert das Zwischenspeichern der zu verschiebenden Fachstunden einschließlich der ursprünglichen Räume, da zu Beginn des Verfahrens noch nicht klar ist, ob der Austausch erstens überhaupt möglich ist und er zweitens, fall er möglich ist, einer Prüfung durch die Ziel-Funktionen standhält. Auch für die Kursstunden müssen die entsprechenden Räume registriert werden. Die ID's der Räume werden in temporären Listen von Zeigern auf Integer-Werte gespeichert. Die Kursstunden selbst sind in jeder einzelnen Kursstunde in der Liste `CStunde::KursStundenL` enthalten und können daher später leicht ermittelt werden.

Ohne die initiale Ersetzung aller auszutauschenden Stunden durch die Leerstunde gehen mögliche Zustands-Übergänge verloren, was eine Einschränkung des Lösungsraumes nach sich ziehen und nur negativ auf die Optimierung wirken kann.
Der nächste Schritt ist die Zufalls-basierte Suche nach den Austausch-Räumen - für die Kursstunden zur Zeit *t1* und für die Fachstunden zur Zeit *t2*. Da erst bei dem Auffinden des letzten der sechs Räume klar ist, ob der Austausch vorgenommen werden kann, müssen die Zwischenergebnisse der Raum-Suche gespeichert werden. Durch dieses vorgeschaltete Sammeln der Räume ist es theoretisch möglich, daß für die Stunden *s2a*, *s2b* und *s2c* ein und der selbe Raum ausgewählt wird. Aus diesem Grund muß stets die Menge der bereits registrierten Räume auf das Vorhandensein des vermeintlich freien Raumes untersucht werden. Das Zwischenspeichern der künftigen Räume der Kurs- bzw. Fachstunden wird durch zwei weitere Listen vorgenommen.
Wurde ein kompletter Satz von Räumen gefunden, kann der eigentliche Austausch erfolgen. Dazu werden die Leerstunden in den entsprechenden time-slots gelöscht, es sei denn, die Leerstunde ist identisch mit der Variablen `CPlan::LeerStunde`. In diesem Fall wird der Inhalt des time-slots lediglich mit der Kurs- bzw. Fachstunde ersetzt. Die folgende Abbildung stellt einen möglichen Austausch-Vorgang dar.

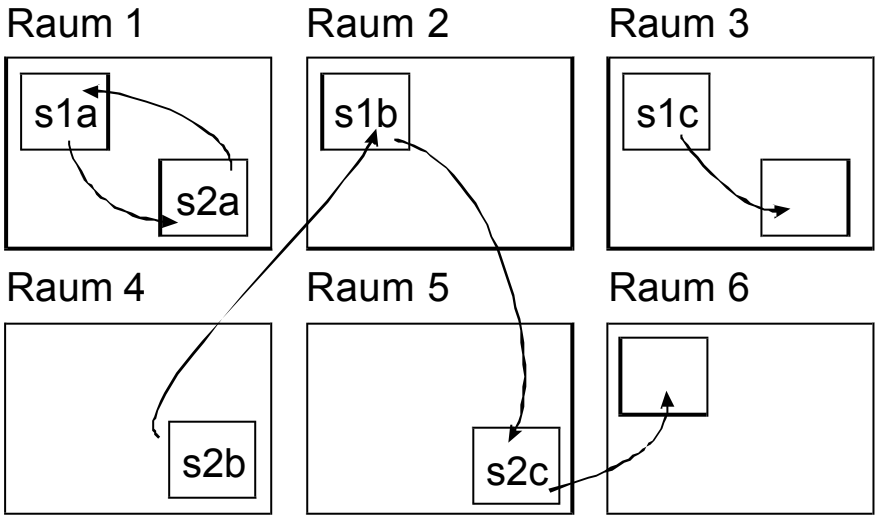


Abbildung 10: Beispiel für einen Austausch-Vorgang in einem Misch-Verfahren

Die Menge der neuen Räume für die Fachstunden ist {1, 5, 3}, die für die Kursstunden {1, 2, 6}. Nur die ehemaligen time-slots der Stunden *s1c* und *s2b* hinterlassen „echte Löcher“. Diese Löcher müssen durch die Erzeugung von neuen `CStunde`-Objekten, die als Leerstunden definiert sind, aufgefüllt werden. Die zusätzliche Erzeugung dieser Stunden wird durch die Löschung der Leerstunden in den künftigen time-slots der Stunden *s1c* und *s2c* wieder aufgehoben.
Falls ein Austausch-Vorgang keine Verbesserung des Planes hervorbringt, muß der ursprüngliche Zustand wieder hergestellt werden.

Bezeichner	`CPlan::Tausch`

Parameter	`int klasse1`
	`int klasse2`
	`int zeit1`
	`int zeit2`
	`CStunde* pS1`
	`CStunde* pS2`

	`bool DoIt`
	`bool manuell`
	`const char OptTyp`

Rückgabe-Typ	`bool`
Kurz-Beschreibung	– Austausch von beliebigen Stunden zweier fester Klassen zu festen Zeiten (`klasse1`, `klasse2`, `zeit1`, `zeit2`) – Verwendung als Misch-Verfahren für die Ziele KLS und KSV und für die direkte Optimierung der Ziele TF, LLS und LSV – `typ` bestimmt das Optimierungs-Ziel und die Optimierungs-Strategie, welche entscheiden, ob der Austausch akzeptiert werden soll – bei `manuell == true` werden die Ziel-Funktionen ignoriert – Rückgabewert informiert bei `DoIt == false`, ob Tausch möglich ist oder nicht und bei `DoIt == true`, ob Tausch erfolgreich vorgenommen wurde oder nicht – `DoIt` bestimmt, ob der Tausch tatsächlich vorgenommen werden soll oder nicht

Die Variablen `klasse1`, `klasse2`, `zeit1` und `zeit2` müssen belegt sein. Ist eine der Variablen `pS1` oder `pS2` gleich Null, ermittelt die Funktion die entsprechenden Stunden selbst. Die Stunden der Zeiten *zeit1* und *zeit2* der Klassen `klasse1` und `klasse2` werden analysiert und die Menge der beteiligten Stunden bei Notwendigkeit vergrößert. Ist eine der Stunden z.B. eine Doppelstunde, erhöht sich die Anzahl der in den Tausch-Vorgang involvierten Stunden von zwei auf vier.
Die Methode ermöglicht beliebige Austausch-Vorgänge und ist damit für zwei Aufgabenbereiche geeignet. Erstens stellt sie eine Verallgemeinerung der Funktion `CPlan::Austausch` dar, da sie z.B. auch den Austausch von Mehrfach-Kursstunden bewältigt. Damit ermöglicht sie komplexere Zustands-Übergänge, und erschließt so weitere Gebiete des Lösungsraumes. Sie könnte also theoretisch `CPlan::Austausch` ersetzen. Die höhere Flexibilität geht allerdings auf Kosten der Geschwindigkeit. Darüber hinaus ist die Wahrscheinlichkeit, Mehrfach-Kursstunden zu verschieben, i. Allg. sehr gering - das zeigt sich auch in der Praxis der manuellen Plan-Erstellung. Daher muß diese Funktion im Optimierungs-Prozeß eher als Ergänzung, denn als treibende Kraft bezeichnet werden. Der Haupt-Anwendungsbereich der Methode liegt in der manuellen Optimierung eines Planes durch den Programm-Nutzer. Wählt man z.B. eine Stunde einer Klasse aus, können durch Iteration über alle übrigen Zeiten alle die Stunden der Klasse ermittelt werden, die mit der gewählten Stunde austauschbar sind. Die Funktion kann also also ein mächtiges Werkzeug für den Nutzer darstellen, da die mühselige Suche nach freien Räumen und Überschneidungs-Problemen entfällt[26].

Bezeichner	`CPlan::KLSMinimierung`
Parameter	`int typ` `int SuchNo`
Rückgabe-Typ	`void`
Kurz-Beschreibung	– Verschiebung von Stunden einer Klasse in nicht belegte time-slots – Verwendung für die direkte Optimierung der Ziele KLS, KSV und LSV – `typ` bestimmt das Optimierungs-Ziel und die Optimierungs-Strategie, welche entscheiden, ob der Austausch akzeptiert werden soll

[26] Diese Möglichkeit der Verwendung wurde in die Software integriert.

	– SuchNo bestimmt, wie oft pro Klasse nach Leerstunde gesucht werden soll – verschiebbar sind: a) Einzel-Fachstunden b) Einzel-Kursstunden

Diese Funktion ist nur bedingt als Misch-Verfahren zu verstehen, da sie vornehmlich für die direkte Optimierung der Ziele KLS und KSV konzipiert ist. Sie sucht nicht belegte Stunden einer Klasse und versucht Unterricht der Klasse in diese zu plazieren. Nicht jede unbelegte Stunde ist automatisch eine Leerstunde. Dies verlangsamt die Optimierung, eröffnet der Funktion jedoch einen breiteren Anwendungsbereich.

5.4. Ziel-Funktionen und deren Variablen

Das Maß für die Stunden-Verteilung wird im folgenden als STUNDEN-PRODUKT bezeichnet und ist wie folgt definiert:

$$StundenProdukt = \prod_{t=0}^{t<Tage} Stunden(t)$$

Tage wird durch den Benutzer festgelegt (normalerweise 5). *Stunden* sei eine Funktion, die die Anzahl der Unterrichtsstunden pro Tag ermittelt (ohne Leerstunden). Es werden also für jeden Tag die Stunden aufsummiert und diese Summen anschließend miteinander multipliziert. Bei gleichmäßiger Verteilung der Stunden über alle Tage erreicht das Stunden-Produkt einen maximalen Wert. Ein Beweis-Ansatz für diese Aussage soll im folgenden dargestellt werden:

Für reelle Zahlen $\alpha_i \in [0,1]$ mit $i = 1, \ldots, n$ und $\sum_{i=1}^{n} \alpha_i = 1$ ist eine Funktion f als konkav definiert, wenn gilt:

$$f(\alpha_1 x_1 + \ldots + \alpha_n x_n) \geq \alpha_1 f(x_1) + \ldots + \alpha_n f(x_n).$$

Wegen der Konkavität des natürlichen Logarithmus ergibt sich unter der Voraussetzung $\alpha_i = \frac{1}{n}$ die Beziehung

$$\ln\left(\frac{x_1 + \ldots + x_1}{n}\right) \geq \frac{1}{n}\left(\ln(x_1) + \ldots + \ln(x_n)\right),$$

wobei Gleichheit gerade für $x_1 = \ldots = x_n$ gegeben ist. Die Ungleichung läßt sich umformen zu

$$e^{\left(n \ln\left(\frac{x_1 + \ldots + x_n}{n}\right)\right)} \geq x_1 \cdot x_2 \ldots \cdot x_n .$$

Für die x_i setze man die jeweilige Anzahl der Stunden pro Tag ein. Bei gleicher Verteilung aller Stunden einer Klasse auf die Unterrichts-Tage wird das Produkt der x_i maximal.

Der optimale Wert wird einmal bei Beginn der Planungs-Phase berechnet und in der Variablen `CKlasse::ProdLimit` bzw. `CLehrer::ProdLimit` abgelegt. Er gilt als Sollwert für die Restriktion Stunden-Verteilung.
Während für einige Variablen das Optimierungs-Ziel klar abgesteckt ist, gilt es für andere Variablen bestmögliche Ergebnisse zu erzielen. Die Beseitigung von Klassen-Leerstunden ist normalerweise eine harte Restriktion und der Sollwert von `LeerStunden` wird daher stets als Null angenommen. Trennungsfehler hingegen lassen einen gewissen Spielraum zu. Zwar kann auch hier ein Sollwert von Null angenommen werden, jedoch läßt sich dieses Ziel nur schwer für alle Klassen erreichen.
Ist `CLehrerStatus::Opti` bzw. `CKlassenStatus::Opti` mit dem Wert `true` belegt, ergibt sich für die Variablen `LeerStunden`, `StundenProd`, `TrennungsFehler` und `SpezialTrennungsFehler` der Wert `0`. Damit liefern die entsprechenden Lehrer bzw. Klassen keinen Beitrag in den globalen Ziel-Funktionen.

Es lassen sich zwei Typen von Optimierungs-Variablen unterscheiden: Parameter, die den Gesamtzustand des Planes beschreiben und Variablen für die einzelnen Klassen und Lehrer. Der Optimierungs-Status von einzelnen Lehrern bzw. Klassen wird durch die bereits in Abschnitt 4.2.9 und 4.2.10 beschriebenen Klassen `CLehrerStatus` bzw. `CKlassenStatus` festgehalten. Die für den gesamten Plan-Zustand verantwortlichen Variablen ergeben sich aus der Summe der einzelnen Variablen für Lehrer und Klassen.
In der folgenden Tabelle sind die Plan-globalen Variablen für die Werte der Ziel-Funktionen und deren Bedeutung aufgeführt.

Ziel	Variablen in `CPlan`	Beschreibung
Klassen-Leerstunden (KLS)	`KLSSum`	– Summe der Leerstunden aller Klassen

Verteilung der Stunden der Klassen (KSV)	`KProdSum`
	`KProdSumLimit`
	`SKProdSum`
		– Summe der Stunden-Produkte aller Klassen
		– Sollwert für `KProdSum`
		– Bestwert des Stunden-Produkts

Trennungsfehler (TF)	`TFSum`
	`TFMax`
	`SpezialTFSum`
	`SpezialTFMax`
		– Summe der Trennungsfehler aller Klassen
		– Maximum der Trennungsfehler aller Klassen
		– Summe der Spezial-Trennungsfehler aller Klassen
		– Maximum der Spezial-Trennungsfehler aller Klassen

Lehrer-Leerstunden (LLS)	`LLSSum`
	`LLSMax`
	`SLLSSum`
		– Summe der Leerstunden aller Lehrer
		– Maximum Leerstunden aller Lehrer
		– Bestwert der Leerstunden-Summe aller Lehrer

Verteilung der Stunden der Lehrer (LSV)	`LProdSum` `LProdSumLimit`	– Summe der Stunden-Produkte aller Lehrer – Sollwert für `LProdSum`
KSV, LLS	`T`	– Optimierungs-Parameter für verschiedene Strategien (siehe 5.5)

Tabelle 7: Plan-globale Optimierungs-Variablen

Verantwortlich für die Berechnung der Ziel-Funktionen sind die Methoden `CPlan::KStatus` und `CPlan::LStatus`, die jeweils die Werte für eine Klasse bzw. für

einen Lehrer ermitteln und die Methode `CPlan::GlobalParas`, die Aufsummierungen und Bestimmungen von Maximal-Werten über alle Lehrer und Klassen vornimmt.

5.5. Optimierungs-Strategien

Die wohl älteste und am weitesten verbreitete Methode für die computergestützte SP-Erstellung ist Tabu Search (TS). TS wird zu den sogenannten greedy-Algorithmen gezählt, welche im Verlauf einer Optimierung stets nur Schritte akzeptieren, die einen Vorteil erbringen und besitzt damit Charakteristiken klassischer Hill-Climbing-Verfahren. Bezogen auf das Lösungsraum-Modell werden also ausschließlich Zustands-Übergänge zu besseren Werten der Ziel-Funktion hin zugelassen. Daher tritt bei diesem Verfahren auch das übliche Problem der Konvergenz gegen lokale Extrema auf. Zur Vermeidung bzw. Abschwächung dieses Effekts werden bestimmte Techniken angewandt, die ein weiträumigeres Durchstreifen des Suchraumes ermöglichen.
Eine speziell für das timetabling entwickelte Technik ist die der sogenannten Tabu-Listen. Dieses Konzept versucht die Wiederholung bestimmter Zustands-Übergänge bzw. das Verharren eines Planes in einem Zyklus von Übergängen zu vermeiden. Dazu wird während der Optimierung eine Liste mit bestimmten Attributen definiert, die den Plan für die entsprechenden Zustandsänderungen sperrt. Als Beispiel für ein solches Attribut möge man sich den Repräsentanten einer bestimmten Unterrichts-Stunde vorstellen.
Zustandsänderungen, an denen dieses Stunde beteiligt ist, werden nun blockiert. Der Erfolg dieser Strategie hängt maßgeblich von der jeweiligen Problemstellung und den Erfahrung bei Optimierungs-Test ab. Aus Zeitgründen wurde auf die Implementierung von Tabu-Listen vorerst verzichtet.
Weiterhin findet eine Gruppe von generischen Techniken wie Simulated Annealing oder Sintflut-Verfahren Anwendung, für die bereits in Abschnitt 4.4.1.1 eine kurze Beschreibung vorgenommen wurde. Da diese Strategien i. Allg. leicht zu implementieren sind, wurden sie kurzfristig als Module mit Prototyp-Charakter während der Entwicklung zu Test-Zwecken in die Software integriert.

5.5.1. Tabu Search

Das Grundprinzip des TS läßt sich auf einfachstem Wege mit den Misch-Verfahren realisieren. Nach jedem Zustandsübergang wird überprüft, ob eine Verbesserung der Optimierungs-Ziele erreicht wurde. Ist dies nicht der Fall ist, wird der ursprüngliche Plan-Zustand wieder hergestellt. Der Frage, ob ein Übergang ohne meßbares Voranschreiten der Optimierung eine „Verbesserung“ darstellt, kommt entscheidende Bedeutung zu. Auf das Beispiel der Klassen-Leerstunden bezogen können nach dem Übergang von Zustand 1 zu Zustand 2 folgende drei Konstellationen für die Summe der Leerstunden aller Klassen (KLSSum) auftreten:

1. $KLSSum_1 > KLSSum_2$
2. $KLSSum_1 = KLSSum_2$
3. $KLSSum_1 < KLSSum_2$

Fall 3 repräsentiert die angestrebte Verbesserung des Optimierungs-Ziels. Fall 2 bringt zwar keinen unmittelbaren Vorteil, erzeugt jedoch einen neuen Plan mit veränderten Überschneidungs-Beziehungen. Diese neuen Konstellationen können durch nachfolgende Übergänge bislang nicht erreichbare Plan-Zustände ermöglichen. In Bezug auf das Lösungsraum-Modell klassischer Hill-Climbing-Verfahren entspräche dieser Fall entweder einem horizontalen Schritt an einem Abhang oder einem Sprung auf einen benachbarten und

möglicherweise höheren Berg. Wird ein höherer Berg erreicht, besteht auch die Chance, bessere Werte für die Ziel-Funktion zu finden. Analog dazu führt eine scheinbar unwirksame Veränderung im Erbgut eines Individuums vielleicht erst dann zu einer besseren Anpassung an die Umwelt, wenn sie mit einer weiteren Mutation kombiniert wird. Und schließlich kehren wir zu dem Lösungsraum-Modell in Abbildung 8 zurück und interpretieren den Zustandsübergang in Fall 2 als Schritt in die „Nähe“ eines weiteren Überganges, der eine echte Verbesserung des Wertes der Klassen-Leerstunden bedeuten könnte. Die Bedingung für die Akzeptanz von Zustands-Übergängen bzgl. eines Zieles Z lautet also: $Z_1 \leq Z_2$. Sollen mehrere Ziele gleichzeitig Optimiert werden, muß dieses Bedingung für alle entsprechenden Werte-Änderungen erfüllt sein.

5.5.2. Simulated Annealing

Simulated Annealing (SA) nimmt die Auswahl von Optimierungs-Schritten bzw. Übergängen auf der Grundlage bestimmter Wahrscheinlichkeiten vor. Übergänge mit schlechteren Optimierungs-Werten werden mit dem Ziel akzeptiert, die Konvergenz gegen lokale Extrema zu vermeiden. Zu Beginn der Optimierung eines Planes kann i. Allg. eine große Menge potentieller Zustands-Übergänge angenommen werden. Jeder dieser Übergänge stellt den Anfang eines Kantenzuges dar, der den Verlauf der Optimierung innerhalb des Lösungsraumes beschreibt. Mit fortschreitender Optimierung verringert sich zwar die Zahl der potentiellen Übergänge für den jeweils nächsten Knoten des Kantenzuges, trotzdem kann jeder neue Zustand über Erfolg oder Mißerfolg der Optimierung entscheiden - ähnlich der Situation, in der ein Suchverfahren in einen Teil-Baum eines Such-Baumes vordringt, der den zu findenden Knoten enthält oder nicht. Ob der Zustand am Ende eines Kantenzuges die Optimierungs-Ziele erfüllt, steht trivialer Weise erst am Ende der Optimierung fest. Schlägt die Optimierung fehl, sollte die Möglichkeit bestehen, in Plan-Zustände zurückzukehren, in denen die falsche Entscheidung bzgl. des eingeschlagenen Pfades durch den Lösungsraum getroffen wurde. Um einen Plan in einen solchen Zustand zurückversetzen zu können, muß eine zeitweilige Verschlechterung der Optimierungs-Parameter in Kauf genommen werden. Beim SA hängt die Auswahl der Übergänge von bestimmten Wahrscheinlichkeiten ab: "Da grundsätzlich in jeder Phase des Verlaufs die Chromosomen mit höherer Fitneß höhere Wahrscheinlichkeiten besitzen, konvergiert das Verfahren im stochastischen Sinne gegen ein Maximum.“[Kin, S. 48][27] „Eine solche Wahrscheinlichkeit ist z.B. gegeben durch

$$p(r) = \frac{1}{1 + e^{\frac{-r}{T}}}$$

wobei T eine beliebige positive Zahl bedeutet.“[Kin, S. 47]

Der Parameter *r* repräsentiert die Differenz der Optimierungs-Werte zwischen zwei Optimierungs-Schritten. Traditionell schwierig gestaltet sich die Wahl der Parameter der Wahrscheinlichkeits-Funktion *p*. *T* sollte im Verlauf der Optimierung langsam ansteigen, während *r* keine allzu großen Werte annehmen sollte, da sonst die Wahrscheinlichkeiten nahe bei 0 oder 1 liegen.

[27] Chromosom = Individuum = Plan

Für die Optimierung der Lehrer-Leerstunden wurde ein Modell auf der Grundlage folgender Schätzungen entwickelt:

- Die Änderung der Lehrer-Leerstunden zwischen zwei Austausch-Vorgängen $LLSSum_2 - LLSSum_1$ beträgt in den meisten Fällen 1 oder 2. Um den Parameter r klein zu halten, wird er bzgl. eines fiktiven Maximalwertes normiert: $r = \frac{LLSSum_2 - LLSSum_1}{5}$.
- Konvergiert der Mittelwert der Lehrer-Leerstunden gegen Null, soll die Wahrscheinlichkeit einen Tausch zu akzeptieren, der eine Verschlechterung der LLS um 1 erzeugt, bei $p \approx 0.01$ liegen. Eine entsprechende Verbesserung der LLS wird mit einer Wahrscheinlichkeit von $p \approx 0.99$ zugelassen.
- Für die Temperatur wird eine lineare Abhängigkeit $T \sim \frac{LLSSum}{L}$ angesetzt, wobei für die Wahrscheinlichkeit, eine Verbesserung der LLS um 1 zuzulassen genau bei $p \approx 0.85$ liegen möge, wenn für den Mittelwert der Leerstunden gilt : $\frac{LLSSum}{L} = 10$.

Die Temperatur-Funktion ergibt sich damit zu $T = 0.0072 \cdot \frac{LLSSum}{L} + 0.043$.

Damit ergeben sich die folgende Kurvenverläufe:

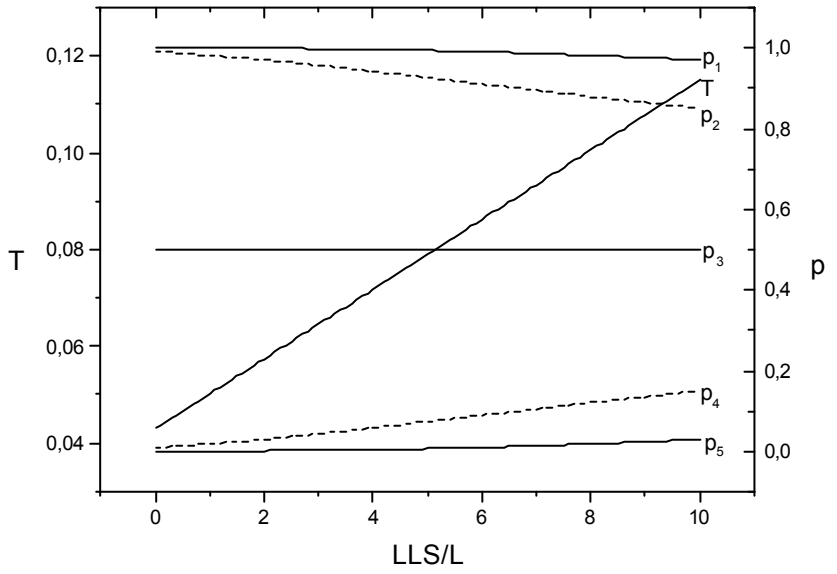


Abbildung 11: Temperatur- und Wahrscheinlchkeits-Funktionen für die Optimierung der LLS mit Simulated Annealing

Das Diagramm enthält die Funktionen $p_1 = p(T, r = 0.2)$, $p2 = p(T, r = 0.1)$, $p_3 = p(T, r = 0)$, $p_4 = p(T, r = -0.1)$ und $p_5 = p(T, r = -0.2)$. Für größere positive Werte für r, liegt die Wahrscheinlichkeit auf einem höheren Niveau (p_1), während sie für kleinere auf einem niedrigeren Niveau mit der Temperatur steigt (p_2). Somit haben Verbesserungen mit höheren Beträgen größere Chancen zugelassen zu werden und Verschlechterungen mit höheren Beträgen geringere Chancen.

5.5.3. Sintflut-Verfahren

Die Grundidee dieses Verfahrens ist, wie beim SA, die Rückkehr zu früheren Plan-Zuständen durch Zulassen von Verschlechterungen, um die Konvergenz der Optimierung in lokalen Extrema zu vermeiden.
Für eine zu maximierende Ziel-Funktion wird ein Schwell- oder Akzeptanz-Wert definiert, oberhalb dessen jede Zustandsänderung zugelassen wird[28]. Veranschaulichen läßt sich diese Methode mit einem Wanderer, der sich in einer bergigen Landschaft stets auf dem Trockenen bewegt, während der Wasserstands-Pegel steigt und nach und nach die Hügel unter der Wasseroberfläche verschwinden. Verkleinerungen des Fitneß-Wertes werden hier also nur bis zu einem bestimmten Schwellwert *T* zugelassen. Da der Schwellwert im Verlaufe des Verfahrens langsam ansteigt, bewegen sich die Fitneß-Werte in immer höheren Bereichen. Dabei bewirkt der Anstieg von *T* letztlich eine immer stärkere Einschränkung des Suchraumes auf Bereiche, in denen sich das absolute oder ein besonders hohes lokales Maximum befinden könnte.
Mehr als beim Simulated Annealing deutet die obige Veranschaulichung dieses Verfahrens auf einen engen Zusammenhang mit klassischen Optimierungs-Problemen und deren Lösungsraum-Modell hin. Problematisch ist vor allem die Forderung nach einer *langsamen* Veränderung des Schwellwertes, denn die Differenz zwischen zwei Schwellwerten kann hier wegen des diskreten Charakters des Problems minimal 1 betragen. Somit besteht bei fortgeschrittener Optimierung stets die Gefahr, daß der Akzeptanz-Wert unterhalb des aktuellen Optimierung-Wertes gesenkt wird (bezogen auf ein *Minimierungs*-Problem).
In Anlehnung an das normale Sintflut-Verfahren wurden daher zwei abgewandelte Formen entwickelt.
Die beiden Varianten unterscheiden sich in der Art der Verschiebung der Akzeptanz-Grenzen - erläutert am Beispiel der LLS-Minimierung. Während sich S1 stets an *LLSSum*, dem *aktuellen* Wert der LLS orientiert, erfolgt die Verschiebung des Schwellwertes in S2 bei Überschreitung von *SLLSSum*, dem bisherigen *Bestwert* der LLS:

Verfahren S1:
Schlechtere Zustände werden auch dann akzeptiert, wenn sie einen bestimmten Schwellwert *T* nicht unterschreiten[29]. In regelmäßigen Abständen wird überprüft, ob der aktuelle Wert der LLS unterhalb von *T* liegt. Ist dies der Fall, wird *T* der aktuelle Wert *LLSSum* zugewiesen. Mit fortschreitender Optimierung werden die Verbesserungen seltener und die Wahrscheinlichkeit sinkt, daß *T* verkleinert wird. Dadurch wird ein langsames Absenken des Schwellwertes erreicht. Die Bedingung für die Akzeptanz eines neuen Zustandes lautet $LLSSum \leq T$.

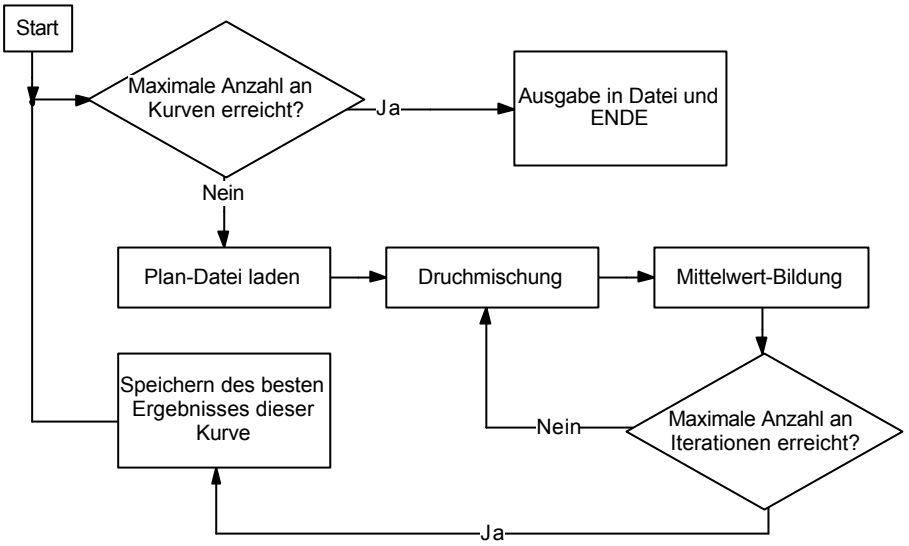
Verfahren S2:
Schlechtere Zustände werden auch dann akzeptiert, wenn sie sich in einem bestimmten Akzeptanz-Intervall befinden. Die untere Intervall-Grenze ist definiert durch *SLLSSum*, dem besten Wert der LLS, der im Verlauf der Optimierung erzielt wurde. Die Intervall-Länge *T* ist ein fester Wert, der vor der Optimierung definiert wird. Die Bedingung für die Akzeptanz eines neuen Zustandes lautet also $LLSSum \leq SLLSSum + T$.
Erreicht der Plan einen Zustand, der den bisherigen Bestwert *SLLSSum* unterschreitet, verschiebt sich die untere Intervall-Grenze und damit das Intervall selbst um die entstandene Differenz der LLS nach unten.

---

[28] Für ein *Minimierungs*-Problem ist nur ein *Unter*schreiten dieses Wertes erlaubt.
[29] T hat unterschiedliche Bedeutung in den Verfahren

5.5.4. Test-Ergebnisse im Vergleich

Für die Tests der Optierungs-Strategien am Beispiel der LLS-Minimierung wurde folgendes Meß-Regime entwickelt:



Die maximale Anzahl der Iterationen beträgt $I = 1000$, die der Kurven $K = 40$.
Der Schritt „Durchmischung“ basiert auf folgenden Aufrufen:

```
Form1->pP->Austausch(S1_LLS,20,20);
for (int i = 0; i < 5; i++)
{
     RandomTausch(S1_LLS);
}
```

`RandomTausch` ist eine Funktion, die Aufrufe der Methode `CPlan::Tausch` mit Zufalls-Parametern kapselt.
Der Algorithmus nimmt K-mal die Optimierung eines bestimmten Ausgangs-Planes vor. Jeweils 1000 Iterationen mit den zugehörigen Bestwerten für `LLSSum` bilden dabei eine Meß-Kurve. Die K Kurven werden überlagert und gemittelt. Die Mittelung soll grundsätzliche Aussagen über das Optimierungs-Verhalten der einzelnen Strategien ermöglichen. Für die praktische Anwendung sind jedoch vor allem die Spitzenwerte bei der Optimierung von Bedeutung. Ein Algorithmus, der viele *gute* Ergebnisse erbringt, ist weniger geeignet als ein Algorithmus mit wenigen *sehr guten* Optimierungs-Werten. Aus diesem Grund ist auch die Häufung der Spitzenwerte für die LLS von Interesse.
Der verwendete Plan wurde „künstlich“ erzeugt. Er enthält einen Querschnitt der möglichen Probleme-Fälle einer Schule wie verschiedene Arten von Sport-Unterricht, Kurse in den höheren Klassen und Mehrfachstunden. Der Plan befand sich in einem Zustand optimierter KSV und minimierter TF. Das Maximum der LLS fand bei diesen Tests keine Berücksichtigung.

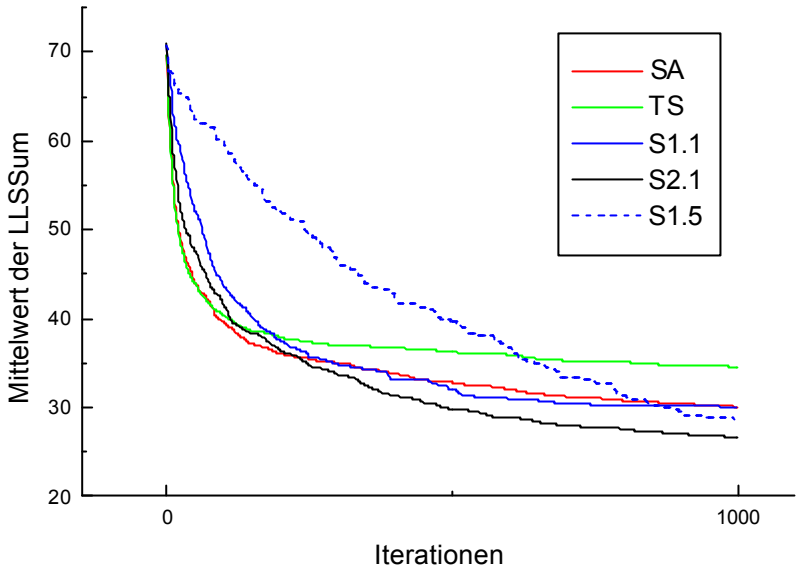


Abbildung 12: Optimierungs-Verhalten von SA, TS, S2 und S1 bei einer Iterationszahl I = 1000

Dargestellt sind die gemittelten Kurven für die Strategien Simulated Annealing, Tabu Search, S2 und S1. Für S1 wurde nach jeder zweiten bzw. fünften Durchmischung[30] eine Neudefinierung des Schwellwertes *T* versucht. Die Länge des Akzeptanz-Intervalls[31] für S2 beträgt 1.

Allgemein läßt sich feststellen, daß Überlegungen zum Optimierungs-Verhalten klassischer Probleme auf das SPP übertragbar sind. So stellt sich Tabu Search erwartungsgemäß als schlechtestes Verfahren heraus. Der „greedy"-Charakter dieses Verfahrens führt offenbar früher als bei allen anderen Strategien zu einer Konvergenz der Optimierungs-Werte. Die Bestwerte der LLS häufen sich folglich bei hohen Werten. Das zeitweilige Zulassen von Verschlechterungen bei allen anderen Verfahren bringt im Gegensatz zum Tabu Search Vorteile für die Optimierung.

In Hinblick auf den Anstieg am Ende der Kurven S1.5 und S2 in Abbildung 12 lassen sich weitere Verbesserungen der Ziel-Funktions-Werte vermuten und es stellt sich die Frage nach dem Kurven-Verhalten bei höheren Iterations-Zahlen. Eine Langzeit-Messung, bei der eine Mittelung von 20 Kurven bei einer Iterationszahl von 4000 für die Strategien S1.5, S1.1, S2.1 und SA vorgenommen wurde, ist in der folgenden Abbildung dargestellt.

[30] S1.*x* ⇒ Neudefinition nach jeder *x*-ten Durchmischung

[31] S2.*x* ⇒ *x* = Akzeptanz-Intervall

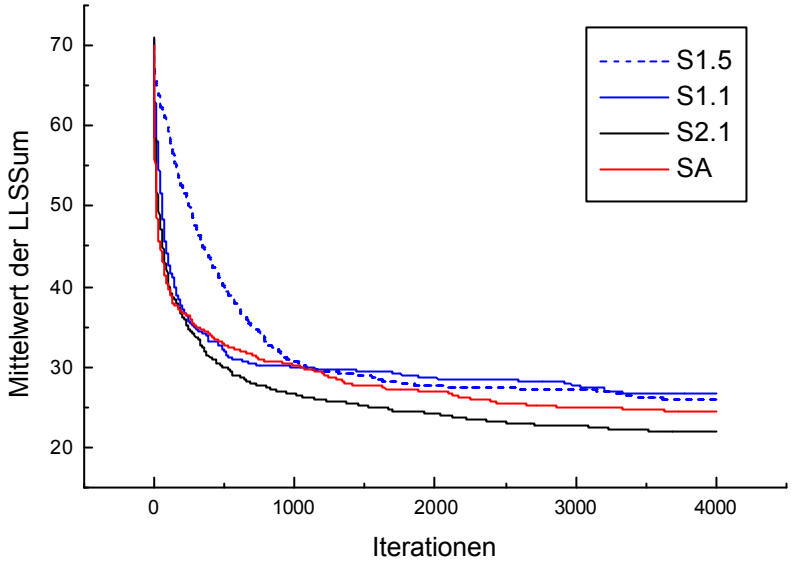


Abbildung 13: Optimierungs-Verhalten von S2.1, S1.5, S1.1 und SA bei einer Iterationszahl I = 4000

S2.1 stellt sich als beste Strategie für die Minimierung der LLS heraus. Als bestes Resultat überhaupt wurde eine Lehrer-Leerstunden-Zahl von 16 ermittelt. Für S1 läßt sich aussagen, daß die verschiedenen Aktualisierungs-Abstände des Schwellwertes *T* nur zu Beginn des Verfahrens einen Einfluß auf die Ergebnisse der Optimierung haben.
SA hängt maßgeblich von den Parametern der Wahrscheinlichkeits-Funktion *p* ab. Daher können aus dem Kurvenverlauf keine allgemeingültigen Aussagen abgeleitet werden und eine weiter Steigerung der Effizienz dieser Strategie ist vorstellbar.

5.5.5. Tests an realen Daten

Das Programm wurde anhand von Daten einer Berliner (Schule1, Grundschule) und einer Potsdamer Schule (Schule2, Gesamtschule) getestet.

Schule1:
- 16 Klassen (1. - 6.)
- 22 Lehrer
- mit Ausnahme von Schwimmen und Sport keine Angaben für Räume notwendig
- keine Kurse
- Nebenbedingungen:
  a) ab der 4. Klassenstufe müssen alle Klassen pro Tag genau einmal Deutsch, einmal Mathematik und einmal Englisch haben
  b) gesperrte Zeiten für einige Lehrer
  c) eindeutig vordefinierter Sport-Unterricht

Für diese Schule konnten erfolgreich Stundenpläne erstellt werden. Bei Einhaltung aller Nebenbedingungen wurde die Zahl der Summe aller LLS auf 18 und das Maximum der LLS auf 3 optimiert.
Als problematisch haben sich Lehrer erwiesen, die als Klassenlehrer in einer der Klassen 1 - 4 eingeteilt waren und zusätzlich Unterricht in höheren Klassenstufen geben mußten. Da diese Lehrkräfte als Klassenlehrer 80 - 90% des Unterrichts ihrer eigenen Klasse gestalten und für

die Klassen keine Leerstunden zugelassen werden, sind sie in den ersten 4 - 5 Stunden pro Tag praktisch nicht verfügbar. Als Ausgleich für das tägliche Frühaufstehen wurde an Schule1 für diese Lehrer eine möglichst Leerstunden-freie Woche gefordert. Dies bedeutet, daß der Unterricht in den höheren Klassenstufen direkt auf die Stunden in der eigenen Klasse zu folgen hat. Daraus ergibt sich praktisch die Forderung nach idealen Stundenplänen für diese Lehrer, da bei sehr guter Stunden-Verteilung keine Leerstunden zugelassen werden dürfen.
Nebenbedingung a) konnte durch zeitliches preschedulings in der Planungs-Phase erfüllt werden. Die Restriktion c) war zum Teil eine Erleichterung, da die zumeist schwierige Organisation des Sport-Unterrichtes entfiel. Allerdings erzeugen die vordefinierten Sport-Stunden folgenschwere Einschränkungen für die Verfügbarkeiten einiger der Sport-Lehrer.

Schule2:

- 23 Klassen (1. bis 10.)
- 29 Lehrer
- 13 Spezial-Räume
- für 6. Klassen 3 Kursstunden pro Woche
- für 7. - 10. Klassen 60 - 70% des Unterrichts in Kursen
- insgesamt 16 Doppelstunden (Kunst-Unterricht)
- Nebenbedingungen:
  a) Integrations-Stunden für einzelne Schüler zu bestimmten Zeiten
  b) Förder-Stunden als Rand-Stunden vordefiniert
  c) Für Referendare an einem Tag pro Woche 1./ 3./ 5. Stunde eines bestimmten Unterrichtes
  d) Schwimmen: Dreifach-Stunden
  e) pro Wochen müssen die 5. - 7. Klassen drei mal zur gleichen Zeit wegen Religions-Unterricht den normalen Unterricht beenden
  f) gesperrte Zeiten für einige Lehrer

Die umfangreichen Nebenbedingungen konnten zwar durch das Daten-Modell in die Planung integriert werden, im aktuellen Entwicklungsstand stellte sich die Software für dieses Problem jedoch als *nicht geeignet* heraus. Nur mit großem manuellen Aufwand konnten die Leerstunden für alle Klassen beseitigt werden. Das mit Tabu Search und manuellen Techniken erzielte Ergebnis für die KSV von 95% des optimalen Wertes ist nicht zufriedenstellend und konnte auch durch Implementierung des Verfahrens S2 für die KSV nicht verbessert werden.
Der Grund für die ungenügende Optimierung ist erstens der hohe Anteil an Kursen und ferner (wie bei Schule1) der Klassenstufen-übergreifende Unterricht einiger Lehrer.

5.5.6. Diskussion

Der Erfolg der Strategien SA, S2 und S1 gegenüber TS deutet auf ein Funktionieren der in Abschnitt 4.4.2 erwähnten Evolutions- bzw. Back-Tracking-Mechanismen hin. Die Strategien lassen für Verschlechterungen der Fitneß-Werte allerdings nur geringen Spielraum. Daher stellt sich die Frage, ob die Verschlechterungen tatsächlich ein „Durchstreifen“ des Lösungsraumes und das Auffinden weiterer lokaler Extrema nach sich ziehen. Diese Unklarheit läßt sich auf die Frage zurückführen, ob die Lösungsräume klassischer Optimierungs-Probleme tatsächlich vergleichbar sind mit dem des SPP. Die Veranschaulichung des Sintflut-Verfahrens anhand eines Wasserstands-Pegels ist z.B. nicht

ohne weiteres auf das SPP übertragbar, denn die klare Trennung zweier Hügel in der Landschaft kann beim SPP wegen den Eigenschaften des Graphen-Modells nicht vorausgesetzt werden. Vielleicht sind es spärlich verteilte Kanten-Züge zu Bereichen mit besseren Voraussetzungen für die Optimierung, deren Erreichen durch eine Verschlechterung der Fitneß wahrscheinlicher wird. Als Analogie möge man sich zwischen halb überschwemmten Hügeln schwer aufzufindende, schmale Brücken vorstellen.

Das Versagen der Algorithmen am Beispiel der Schule2 ist zurückzuführen auf die begrenzten Fähigkeiten der Misch-Verfahren, komplexere Plan-Strukturen als Einzel-, Kurs- und Mehrfachstunden zu verschieben und auszutauschen. Manuelle Techniken für die KSV basieren z.B. auf dem zeitweiligen Zulassen von Klassen-Leerstunden oder dem *gezielten* Beseitigen von Problem-Konstellationen bzw. Überschneidungen. Die Software sollte also durch Mischverfahren erweitert werden, die „intelligentere“ manuelle Techniken simulieren als die bisher entwickelten. Das Zulassen von Leerstunden bei der Optimierung der KSV würde darüber hinaus Optimierungs-Strategien erfordern die *mehr als nur ein Teil-Ziel gleichzeitig* zu kontrollieren in der Lage sind.

6. Zusammenfassung

Für ein an der Praxis orientiertes SPP wurden flexible Datenstrukturen und leistungsfähige Optimierungs-Verfaren entwickelt und implementiert. Die Optimierungs-Algorithmen werden durch ein Modul für die manuelle Planerstellung ergänzt. Unterstützt werden Schulen mit Unterricht im Klassenverband. Die Aufteilung von Klassen in eine begrenzte Anzahl von Kursen und prescheduling von Unterrichtsstunden ist möglich.
Der aktuelle Stand der Programmierung läßt eine Nutzung der Software in der Praxis zu. Der Schritt zu einem kommerziellen Produkt setzt jedoch weitere Arbeit an Rahmenbedingungen wie Programm-Bedienung und Benutzer-Dokumentation voraus.

Die Optimierung basiert auf einem Mutations-Selektions-Verfahren, welches durch verschiedene Optimierungs-Strategien abgerundet werden kann. An dem Optimierungs-Ziel der Lehrer-Leerstunden wurden die Strategien Tabu Search, Simulated Annealing und zwei für das SPP modifizierte Formen des Sintflut-Verfahrens untersucht. Als effizienteste Strategie hat sich die Variante S2 des Sintflut-Verfahrens erwiesen.
Grundlage des Mutations-Selektions-Verfahrens sind Methoden für die Durchmischung eines Planes durch Verschiebung und Austausch von Einzelstunden, den elementarsten Bausteinen und Bausteinen komplexerer Struktur wie Kurs- oder Mehrfachstunden. Die Effizienz der Optimierungs-Verfahren sinkt mit steigender Zahl von Kursen und Mehrfachstunden.

Der in der Einleitung formulierte Anspruch, eine vollautomatische Plan-Erstellung zu entwickeln, mußte revidiert werden. Die Vielfalt möglicher Nebenbedingungen und die individuellen Wünsche an einen Stundenplan sind nur schwer vorherzusehen. Weiterhin ist die vollständige Simulation effizienter menschlichen Fähigkeiten bei der manuellen SP-Erstellung bisher nicht realisierbar. Es hat sich letztlich als günsig erwiesen, die Kontrolle des Optimierungs-Prozesses zum Teil dem Programm-Nutzer zu überlassen.

7. Blick in die Zukunft

Für die Zukunft sind folgende Erweiterungen der Software angedacht:

- „intelligentere“, an der manuellen SP-Erstellung orientierte Misch-Verfahren
- Überprüfung der Anwendbarkeit der Optimierungs-Strategien SA, S2 und S1 auf alle übrigen Optimierungs-Ziele
- Zwischenspeichern von Plan-Ergebnissen während der Optimierung
- Verbesserungen der Benutzer-Schnittstelle hinsichtlich Bediener-Freundlichkeit
- Integration von gymnasialen Kurs-Systemen

Anwendung von Evolutions-Programmen im Unterricht

Die Funktionsprinzipien der natürlichen Evolution sind einfach und überschaubar. Es ist eine naheliegende Vermutung, daß Prinzipien, die das überaus komplexe Ökosystem der Erde hervorzubringen in der Lage sind, sich auch auf vergleichsweise banale Probleme anwenden lassen.
Kenntnisse aus dem Biologie-Unterricht sind hilfreich aber nicht unbedingt notwendig. Programmier-Kenntnisse sollten vorhanden sein.
Anwendungen lassen sich leicht finden. Allerdings sollten es Beispiele sein, die die Vorteile evolutionärer Algorithmen offen zu Tage treten lassen. Dies währe etwa möglich, wenn ein sicheres Durchsuchen des Problemraumes nur über eine kombinatorische Explosion führen würde.
Die Beschäftigung mit dem Thema Evolution kann allgemein ein Gefühl für die Zerbrechlichkeit des *subtilen Gleichgewichts von Konkurrenz, Antagonismus und Symbiose*[32] großer Biotope oder Ökosysteme vermitteln. Eine Wertschätzung der empfindlichen Einzigartigkeit der lebenden Materie, die in einem Optimierungs-Prozeß entstanden ist, dessen Dauer für den menschlichen Verstand kaum faßbar ist, kann ein kleinen Beitrag zum Thema Umweltschutz liefern.

[32] S. Lem: „Der Flop“, Verlag Volk und Welt, Berlin 1986.

8. Literatur

[Mic]
Z. Michalewicz: „Genetic Algorithms + Data Structures = Evolution Programs“, New York, Springer-Verlag Berlin Heidelberg, 1996.

[Bäc]
T. Bäck: „Evolutionary Algorithms in Theory and Practice“, Oxford University Press, 1996.

[Kin]
W. Kinnebrock: „Optimierung mit genetischen und selektiven Algorithmen“, München, Oldenbourg Verlag GmbH, 1994.

[Stro]
B. Stroustrup: „Die C++ Programmiersprache“, Addison-Wesley, 1992.

[Mey]
S. Meyers: „Effektiv C++ programmieren: 50 Möglichkeiten zur Verbesserung Ihrer Programme“, Addison-Wesley, 1995.

[Gar]
B. Gardé: „Ohne Computer wird es nicht gehen“, Computer und Schule, **33/ 1999**, 20 - 23.

[Eve]
S. Even, A. Atai, A. Shamir: „On the complexity of timetable and multicommodity flow problems“, J. Comput., **Vol. 5**, No. 4, December 1976.

[Coo]
B. Cooper, J. H. Kingston: „The complexity of timetable construction problems“, Lecture Notes in Computer Science, **1153**, 283 - 295.

[Erb]
W. Erben, J. Keppler: „A Genetic Algorithm Solving a Weekly Course-Timetabling Problem“, Lecture Notes in Computer Science, **1153**, 198 - 211.

[Bar]
V. A. Bardadym: „Computer-Aided School and University Timetabling: The New Wave“, Lecture Notes in Computer Science, **1153**, 22 - 45.

[Wei]
J. Weingärtner, „Kurz - Beleg zum Seminar Optimierung und Simulation“, Universität Potsdam 1997, unveröffentlicht.

[Dum]
Dumke, „UML - Tutorial“, Otto-von-Guericke-Universitdt Magdeburg, http://ivs.cs.uni-magdeburg.de/~dumke/UML/index.htm.

[Sch]
H. Schneider, „Lexikon der Informatik und Datenverarbeitung“, R. Oldenvourg Verlag München Wien 1991.

[Dow]
K. A. Dowsland: „Off-the-Peg or Made-to-Measure? Timetabling and Scheduling with SA and TS“, Lecture Notes in Computer Science, **1408**, S. 37 ff.

[Cra]
F. Cramer: „Chaos und Ordnung“, Deutsche Verlags-Anstalt GmbH, Stuttgart 1989.

[Lug]
Lugowski, Weinert: „Grundzüge der Algebra“, B.G. Teubner Verlagsgesellschaft, Leipzig 1968.

[Ste]
A. und K. Steinmüller: „Charles Darwin - vom Käfersammler zum Naturforscher“, Verlag Neues Leben, Berlin 1985.

Danksagung

Mein Dank gilt besonders der Potsdamer Firma CODIE SOFTWARE PRODUCTS für die Bereitstellung der Entwicklungs-Umgebung und der Web-Publishing-Software.
Für viel Geduld und wertvolle Ratschläge danke ich weiterhin Herr Dr. Straubel von der Berliner Ingenieursgesellschaft REUS.

Erklärung

Ich versichere, daß ich die schriftliche Hausarbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem einzelnen Fall unter genauer Aangabe der Quelle deutlich als Entlehnung kenntlich gemacht.

(Unterschrift) (Ort, Datum)