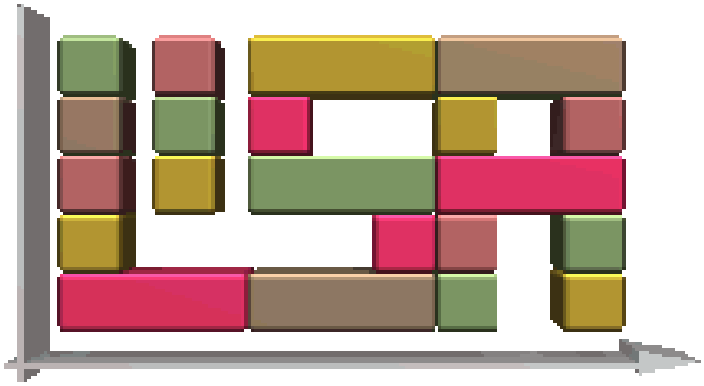


LiSA - A Library of Scheduling Algorithms
Handbuch zu Version 3.0

Michael Andresen, Heidemarie Bräsel, Frank Engelhardt, Frank Werner
Fakultät für Mathematik
Otto-von-Guericke Universität Magdeburg

Abstrakt

LiSA - A Library of Scheduling Algorithms ist ein Softwarepaket zur Lösung deterministischer Schedulingprobleme, insbesondere von Shop-Problemen, die in der Form $\alpha \mid \beta \mid \gamma$ gegeben sind, wobei α die Maschinenumgebung, β zusätzliche Restriktionen für die Aufträge und γ die Zielfunktion beschreibt. Die Entwicklung von LiSA wurde durch Förderung der Projekte *Lateinische Rechtecke in der Schedulingtheorie (1997-1999)* und *LiSA - A Library of Scheduling Algorithms (1999-2001)* durch das Kultusministerium des Landes Sachsen-Anhalt ermöglicht. Ergebnisse von Diplomarbeiten und Dissertationen in unserer Forschungsgruppe sowie von Praktika der Studierenden sind in das Softwarepaket eingeflossen.

Das Handbuch enthält alle notwendigen Informationen, um mit LiSA arbeiten zu können: Lizenzbedingungen, technische Voraussetzungen, mathematisches Basiswissen zu den in LiSA verwendeten Modellen, Beschreibung der Algorithmen und des verwendeten Fileformats, Beispiel zur Arbeit mit Lisa, Anleitungen zum Einfügen eigener Algorithmen und zum automatisierten Algorithmenaufruf.

Das Handbuch ist auch als html-File auf der LiSA-Homepage verfügbar, die Erläuterungen zu den Algorithmen und zu den LiSA-Komponenten sind als Hilfedateien in die Software eingebunden.

Homepage: http://lisa.math.uni-magdeburg.de

Kapitel 1

Allgemeine Informationen

1.1 Was ist LiSA? - Einführung und Übersicht

LiSA - A Library of Scheduling Algorithms ist ein Softwarepaket zur Lösung von deterministischen Shop-Schedulingproblemen. In einem Shop-Schedulingproblem soll eine Menge von Jobs (Aufträgen) unter bestimmten zusätzlichen Bedingungen so auf einer Menge von Maschinen bearbeitet werden, dass eine Zielfunktion optimal wird. Alle Parameter sind bekannt und fest vorgegeben. In der Literatur werden solche Probleme gewöhnlich durch ein Tripel $\alpha \mid \beta \mid \gamma$ beschrieben, wobei α die Maschinenumgebung, β zusätzliche Bedingungen für die Aufträge und γ die Zielfunktion beschreibt. Eine zulässige Lösung eines Shopproblems wird als Plan und der zugehörende zeitliche Ablauf als Zeitplan oder Schedule bezeichnet. Pläne und Schedules werden durch Matrizen beschrieben und in LiSA durch azyklische Digraphen bzw. Ganttdiagramme visualisiert. Da die meisten Shopprobleme schwierig zu lösen sind, enthält LiSA eine Vielzahl von konstruktiven und iterativen Heuristiken.

Will ein Nutzer ein Schedulingproblem mit LiSA lösen, hat er unter Verwendung der grafischen Nutzeroberfläche zuerst den Problemtyp in der $\alpha \mid \beta \mid \gamma$ Notation und die Anzahl der Aufträge und Maschinen einzugeben. Die Bearbeitungszeiten und die restlichen Parameter des Inputs lassen sich per Hand eingeben oder können mit einem Zufallszahlengenerator erzeugt werden. Es ist auch möglich, alle diese Daten aus einer xml-Datei einzulesen. Sobald ein Input vollständig in LiSA vorhanden ist, werden alle Algorithmen zur Nutzung freigegeben, die dieses Problem exakt oder näherungsweise lösen. Nach Anwendung eines Algorithmus wird der Schedule in einem Ganttdiagramm visualisiert, das auftrags- bzw. maschinenorientiert ausgewählt werden kann. Liegt eine erste Lösung vor, werden die iterativen Algorithmen zur Nutzung freigegeben. Da in den Algorithmen in den meisten Fällen noch Parametersetzungen möglich sind, hat der Nutzer eine Vielzahl von Algorithmen zur Verfügung. Die Lösung kann in einer XML-Datei abgespeichert werden.

LiSA enthält einige Extras, wie z.B. den Abruf des Komplexitätsstatus (mit Literaturquelle) eines Problems. Hier wird die Datenbank der Osnabrücker Kollegen zur Komplexität von Schedulingproblemen genutzt. Weiter sind Manipulationen des Ganttdiagramms möglich, so dass der Nutzer selbst in die Konstruktion eines Schedules eingreifen kann.

LiSA hat einen modularen Aufbau, in dem jeder Algorithmus auch extern nutzbar ist. So ist es möglich, einen Algorithmus mit einer Kommandozeile aufzurufen oder ihn in einen au-

tomatisierten Algorithmenaufruf einzubinden. Bei einem solchen automatisierten Algorithmenaufruf lassen sich mit Eingabe eines Problemtyps und der Parameter eine einzugebende Anzahl von Beispielen mit allen für diesen Problemtyp anwendbaren Algorithmen lösen. LiSA legt für jedes Beispiel eine Liste der durch die Algorithmen erzeugten Lösungen an, die wiederum direkt aus der grafischen Nutzeroberfläche aufgerufen werden kann. Aus der bei der Rechnung erzeugten log-Datei lassen sich die Ergebnisse in eine mit Excel kompatible Datei filtern, so dass eine schnelle Auswertung möglich ist. Darüber hinaus können mit diesem Konzept auch hybride Algorithmen konstruiert werden.

Durch den modularen Aufbau ist es für den Nutzer einfach, eigene Algorithmen in LiSA einzufügen. Dazu braucht er neben dem C++ Quelltext eine XML-Datei, die seinen Algorithmus mit der grafischen Nutzeroberfläche verbindet, damit LiSA bei Eingabe des von ihm betrachteten Problems den Algorithmus automatisch zur Nutzung freigibt. Eine Hilfedatei für den neuen Algorithmus ermöglicht anderen Nutzern die Anwendung des neuen Verfahrens.

Dieses Handbuch ist folgendermaßen aufgebaut:

Kapitel 1 enthält neben dieser Einführung eine Übersicht über das LiSA-Team sowie die Systemanforderungen zur LiSA-Nutzung und die Lizenzbedingungen.

Die in LiSA verwendeten Bezeichnungen, die Klassifizierung der Probleme, das verwendete Block-Matrizen-Modell mit den grundlegenden Basisalgorithmen und eine Übersicht zu den in LiSA enthaltenen Algorithmen sind in Kapitel 2 enthalten. Hier wird auch auf das in LiSA verwendete File-Format eingegangen.

Die folgenden drei Kapitel beschreiben die Eingabe eines Problems (Kapitel 3) , die in LiSA enthaltenen Algorithmen (Kapitel 4), unterteilt in universell anwendbare exakte Algorithmen, universell anwendbare konstruktive und iterative Heuristiken sowie spezielle Algorithmen und die Ausgabe der Ergebnisse (Kapitel 5).

Das sechste Kapitel enthält die Extras, begonnen wird mit der Beschreibung zusätzlicher interner Programmbausteine, wie z.B. die Ermittlung des Komplexitätsstatus eines Problems, die Manipulation eines Schedules oder die Reduzierbarkeitsalgorithmen.
Weiter wird das Einfügen eigener Algorithmen und der automatisierte Algorithmenaufruf beschrieben.

Kapitel 8 enthält einige anschauliche Beispiele zur Arbeit mit LiSA.

Das Handbuch schließt im Anhang mit einer XML-Referenz, den GNU-Lizenzbedingungen und einer Literaturübersicht.

1.2 Systemanforderungen und Lizenz

LiSA ist lizensiert unter GPL (GNU General Public License), die Bedingungen für diese Lizenz sind im Anhang enthalten.

LiSA wurde primär für Unix Systeme entwickelt. Zur Compilierung ist ein Standard C++ Compiler und Tcl/Tk 8.0 oder höher erforderlich, wir empfehlen den gcc 3.2 (oder höher) und Tcl/Tk 8.4.
Die Version 2.3 wurde komplett unter SuSE Linux 8.1 entwickelt. Die Weiterentwicklung zur Version 3 nutzte SuSE Linux 9.2 und Tcl/Tk 8.4. Erfolgreiche Tests der Version 3.0 wurden auf folgenden Systemen durchgeführt:

- *SuSE 9.2 unter Nutzung von gcc 3.4.4 und Tcl/Tk 8.4*;
- *SuSE 9.3 unter Nutzung von gcc 3.3.5 und Tcl/Tk 8.4*;
- *SuSE 10.3 unter Nutzung von gcc 3.3.5 und Tcl/Tk 8.4*;
- *Solaris 9.0 unter Nutzung von gcc 3.4.4 und Tcl/Tk 8.4*;
- *Fedora 4.0 unter Nutzung von gcc 4.0.2 und Tcl/Tk 8.4*;
- *Debian 5.0 unter Nutzung von gcc 4.4.1 und Tcl/Tk 8.5*;

Frühere LiSA-Versionen konnten erfolgreich auf folgenden Rechnern compiliert werden:

- *SunOS 5.6/5.7 (sparcSTATION und ULTRAsparc)*;
- *IRIX 6.4*;
- *HP-UX 09.05/10.10/10.20*;
- *AIX 4.2*;
- *SuSE 6.1, 8.0, 8.1*;
- *RedHat 5.1.*
- *Solaris 8.*

Unter Windows lässt sich LiSA mit Hilfe der Cygwin-Plattform compilieren, die eine komplette Unix-ähnliche Umgebung bereitstellt. Speziell für Windows-Systeme bieten wir aber auch einen automatischen Installer an, der LiSA auf Windows 2000, Windows XP, Windows Vista und Windows 7 installiert, ohne dass weitere Software benötigt wird.
Hinweise zur Compilierung von LiSA auf verschiedenen Plattformen lassen sich der Datei `INSTALL` entnehmen, die dem Quellcodepaket beiliegt.

1.3 Das LiSA-Team

Forschungsgruppen an Universitäten sind über längere Zeiten selten homogen zusammengesetzt. Der folgende Überblick enthält die Namen der meisten an der Entwicklung von LiSA beteiligten Wissenschaftler und Studierenden und ihre hauptsächlichen Arbeitsgebiete:

Heidemarie Bräsel: Initiatorin der Projekte, Leiterin des LiSA-Teams und Betreuerin der Graduierungsarbeiten (1997-2009)
Thomas Tautenhahn: Verantwortlich für die Effizienz der Datenstrukturen und Algorithmen, Betreuung von Studierenden in der Programmierung (1997-2000), Habilitation 2002
Per Willenius: Verantwortlich für die grafische Oberfläche von LiSA, der entsprechenden Algorithmen und für das fehlerfreie Miteinander aller Programmmodule, Betreuung von Studierenden in der Programmierung (1997-2001), Dissertation 2000
Martin Harborth: Verantwortlich für das Komplexitätsmodul im LiSA-Hauptprogramm, Be-

treuung von Studierenden in der Programmierung (1997-1999), Dissertation 1999
Lars Dornheim: Verantwortlich für die Kompatibilität von LiSA auf verschiedenen Rechnerkonfigurationen unter verschiedenen Betriebssystemen (1998-1999)
In der ersten Zeitperiode waren die folgenden Studierenden einbezogen:
Ines Wasmund: Visualisierung der Schedules in Ganttdiagrammen
Andreas Winkler: Verschiedene Nachbarschaftssuchverfahren
Marc Mörig: Matching Algorithmen für open-shop Probleme, Reduzierbarkeitsalgorithmen
Christian Schulz: Shifting-Bottleneck Heuristik für das job-shop Problem
Manuela Vogel: Heuristik für das flow-shop Problem.

Kurzzeitig arbeiteten die folgenden Studierenden im Projekt mit:
Holger Hennes, Birgit Grohe, Christian Tietjen, Carsten Malchau und *Tanka Nath Dhamala* (Sandwichstipendiat, Dissertation 2002). In einem Computerpraktikum 2001 fügten *Thomas Klemm*, *Andre Herms, Jan Tusch*, *Ivo Rössling*, *Marco Kleber* und *Claudia Isensee* neue Algorithmen in LiSA ein.

Im Jahr 2002 bereiteten *Lars Dornheim, Sandra Kutz, Marc Mörig und Ivo Rössling* LiSA zur kooperativen Entwicklung vor. Die Modularität der Software wurde entscheidend verbessert. Viele Fehler wurden behoben, so dass die Software stabiler läuft. Ein spezieller LiSA-Server wurde zur Kommunikation zwischen Entwicklern und Nutzern von LiSA eingerichtet, ein Versionsmanagementsystem und ein Fehlermeldesystem in die Arbeit einbezogen. Das Konzept des Einfügens eigener Algorithmen wurde verbessert und vereinfacht. Eine neue Homepage wurde gestaltet. Die LiSA-Version 2.3 war fertig. Die Studierenden wurden für ihre Arbeit an LiSA auf der Studentenkonferenz der DMV-Tagung 2002 in Halle mit dem Sonderpreis der Jury ausgezeichnet.

Wesentlichen Anteil an der LiSA-Version 3 haben Marc Mörig, Jan Tusch, Mathias Plauschin und Frank Engelhardt. Fortschritte in LiSA - Version 3 bestehen im folgenden:
- Das Dateiformat ist von *Jan Tusch* auf .xml umgestellt worden, er hat auch die genetischen Algorithmen implementiert;
- Der Algorithmenaufruf ist von *Marc Mörig*, weiter von *Mathias Plauschin* und von *Frank Engelhardt* automatisiert worden. Sie haben auch die Filterung der Ergebnisse in eine mit Excel kompatible Datei weiterentwickelt, erstmals von *Andre Herms* in Perl implementiert;
- Es gibt für die Windows-Versionen einen automatischen Installer von *Mathias Plauschin*, weiterentwickelt von *Frank Engelhardt*. Bei der Anwendung des Installers wird keine weitere Software benötigt.
Seit vier Jahren arbeitet auch *Frank Werner* im LiSA-Team mit, hauptsächlich an Veröffentlichungen unter Nutzung der Algorithmen in LiSA. Er ist verantwortlich für die englische Version dieses Handbuchs.

Kapitel 2

Basiswissen

2.1 Definitionen und Bezeichnungen

In einem *Shop-Schedulingproblem* soll eine Menge von *Jobs* auf einer Menge von *Maschinen* in einer festgelegten *Maschinenumgebung* unter verschiedenen *zusätzlichen Restriktionen* so bearbeitet werden, dass eine *Zielfunktion* optimiert wird. Das Problem heisst *deterministisch*, wenn alle Parameter bekannt und fest vorgegeben sind. Eine Vielzahl von Optimierungsproblemen, die eine optimale Anordnung von Aktivitäten unter beschränkten Ressourcen suchen, lassen sich als Schedulingprobleme modellieren.
Die folgende Tabelle 2.1 enthält die grundlegenden Bezeichnungen, die in LiSA benutzt werden:

Bezeichnungen	
n, m	Anzahl der Jobs bzw. Anzahl der Maschinen
$\{A_1, \ldots, A_n\}$	Menge der zu bearbeitenden Jobs
$I = \{1, ..., n\}$	Menge der Indizes der Jobs
$\{M_1, \ldots, M_m\}$	Menge der Maschinen, die die Jobs bearbeiten
$J = \{1, ..., m\}$	Menge der Indizes der Maschinen
$p_{ij} \geq 0$	Bearbeitungszeit für Job A_i auf Maschine M_j
$PT = [p_{ij}]$	Matrix der Bearbeitungszeiten
(ij)	Operation, d.h. die Bearbeitung von A_i auf M_j
SIJ	Menge der Operationen (ij)
u_i, v_j	Anzahl der Operationen von A_i bzw. auf M_j
c_{ij}	Fertigstellungszeit der Operation (ij)
C_i	Fertigstellungszeit des Jobs A_i
$C = [c_{ij}]$	Matrix der Fertigstellungszeiten
r_i, d_i	Bereitstellungszeit bzw. Fälligkeitstermin des Jobs A_i
w_i	Gewicht des Jobs A_i
$L_i = C_i - d_i$	Terminabweichung für Job A_i
$T_i = \max\{0, C_i - d_i\}$;	Verspätung von Job A_i
$U_i = \begin{cases} 0, & \text{wenn } C_i \leq d_i \\ 1, & \text{sonst} \end{cases}$	$\sum U_i$ zählt die verspäteten Jobs

Tabelle 2.1: Grundlegende Bezeichnungen für deterministische Schedulingprobleme

Für ein Schedulingproblem mit mehr als einer Maschine werden folgende Reihenfolgen eingeführt:
Die *technologische Reihenfolge des Jobs* A_i ist die Reihenfolge der Maschinen, auf denen dieser Job bearbeitet wird: $M^i_{j_1} \to M^i_{j_2} \to \ldots \to M^i_{j_{u_i}}$, wobei $j_1, \ldots, j_{u_i}$ eine Permutation der Zahlen $1, \ldots, u_i$ ist.
Die *organisatorische Reihenfolge auf der Maschine* M_j ist die Reihenfolge der Aufträge auf dieser Maschine: $A^j_{i_1} \to A^j_{i_2} \to \ldots \to A^j_{i_{v_j}}$, wobei $i_1, \ldots, i_{v_j}$ eine Permutation der Zahlen $1, \ldots, v_j$ ist. Wenn es nicht zu Mißverständnissen führt, wird auf die oberen Indizes verzichtet.

2.2 Klassifizierung deterministischer Schedulingprobleme

In LiSA wird die $\alpha \mid \beta \mid \gamma$ Klassifizierung für deterministische Schedulingprobleme verwendet, eingeführt von Graham et al. [15], wobei

- α die Maschinenumgebung,
- β Jobcharakteristika und weitere Bedingungen und
- γ die Zielfunktion beschreibt.

In LiSA wird die Klassifizierung nicht nur zur Beschreibung eines zu lösenden Problems sondern auch zur Bestimmung des Komplexitätsstatus des betrachteten Problems benutzt. Die Tabellen 2.2, 2.3 und 2.4 geben eine Übersicht möglicher Parameter eines $\alpha \mid \beta \mid \gamma$ Problems, ohne Garantie auf Vollständigkeit. Im Unterschied zur Literatur, in der in einem job-shop Problem das mehrmalige Bearbeiten eines Jobs auf einer Maschine erlaubt ist, besteht für LiSA für alle Shopprobleme die Voraussetzung, dass jeder Job höchstens einmal auf jeder Maschine bearbeitet wird (klassischer Fall).
Das β-Feld kann keinen, einen oder mehrere Parameter aus $\{\beta_1, \ldots, \beta_5\}$ enthalten. Hier sind noch eine Vielzahl anderer Bedingungen möglich, wie z.B.
- *no-wait* : Wartezeiten zwischen zwei aufeinanderfolgenden Operationen des gleichen Jobs sind nicht erlaubt.
- *no-idle* : Stillstandszeiten zwischen zwei aufeinanderfolgenden Jobs auf der gleichen Maschine sind nicht erlaubt.
- $p_{ij} \in \{1, 2\}$: Es sind nur die Bearbeitungszeiten aus $\{1, 2\}$ erlaubt.

Jede der Zielfunktionen $F(C_1, \ldots, C_n)$ aus Tabelle 2.4 ist *regulär*, d.h. wenn $C^*_i \geq C_i \; \forall \; i \in I$ erfüllt ist, dann gilt $F(C^*_1, \ldots, C^*_n) \geq F(C_1, \ldots, C_n)$.
Eine nichtreguläre Zielfunktion ist z.B. die Minimierung der Strafkosten, die bei zu früh oder zu spät fertiggestellten Jobs entstehen.

Es gibt eine Vielzahl weiterer Bedingungen der Bearbeitung der Jobs, z.B. ist ein flexibles flow-shop Problem (FFS) eine Kombination aus flow-shop- und Parallelmaschinenproblem. Sind technologische Reihenfolgen nur teilweise vorgegeben, handelt es sich um ein gemischtes Shopproblem. Weiter gibt es Schedulingprobleme mit Ressourcenbeschränkungen, Batchingprobleme und viele andere.

Maschinenumgebung $\boldsymbol{\alpha = \alpha_1\alpha_2}$	
$\alpha_1 \in \{1, P, Q, R\}$	Jeder Job besteht aus genau einer Operation, die beliebig auf einer der gegebenen Maschinen ausgeführt werden kann.
$\alpha_1 = 1$	Es gibt nur eine Maschine, also gilt: $p_{i1} = p_i$.
$\alpha_1 = P$	Jeder Job wird exakt einmal auf einer von m identischen parallelen Maschinen bearbeitet, d.h. $p_{ij} = p_i$ gilt.
$\alpha_1 = Q$	Jede der m parallelen Maschinen hat die gleiche *Geschwindigkeit* s_j, d.h. $p_{ij} = p_i/s_j$ gilt.
$\alpha_1 = R$:	Die Geschwindigkeiten s_{ij} für die Bearbeitung der Operation (ij) hängt sowohl vom Job A_i als auch der Maschine M_j ab, d.h. $p_{ij} = p_i/s_{ij}$ gilt.
$\alpha_1 \in \{O, F, J\}$	Jeder Job wird auf jeder Maschine genau einmal oder höchstens einmal bearbeitet (klassischer Fall).
$\alpha_1 = O$	open-shop Problem: Die technologischen und organisatorischen Reihenfolgen sind beliebig wählbar
$\alpha_1 = J$	job-shop Problem: Die technologischen Reihenfolgen sind fest und die organisatorischen Reihenfolgen sind frei wählbar.
$\alpha_1 = F$	flow-shop Problem: Die technologischen Reihenfolgen sind fest und identisch für jeden Job, w.l.o.g.: $M_1 \to M_2 \to \ldots \to M_m$. Die organisatorischen Reihenfolgen sind frei wählbar.
$\alpha_2 \in \{\circ, c\}$	Charakterisierung der Anzahl der Maschinen
$\alpha_2 = c$	Die Anzahl der Maschinen m ist konstant, $m = c$.
$\alpha_2 = \circ$	Die Anzahl der Maschinen ist variabel, i.e. ein Teil der Eingabeinstanz.

Tabelle 2.2: Parameter der Maschinenumgebung α

Charakterisierung der Jobs und zusätzliche Bedingungen $\{\beta_1, \ldots, \beta_5\}$	
$\beta_1 \in \{\circ, pmtn\}$	Unterbrechung von Operationen
$\beta_1 = \circ$	Unterbrechung ist nicht erlaubt.
$\beta_1 = pmtn$	Unterbrechung ist erlaubt, i.e. die Bearbeitung eines Jobs auf einer Maschine kann gestoppt und später fortgesetzt werden.
$\beta_2 \in \{\circ, prec, outtree, intree, tree, chain\}$	Die Vorrangbedingung $A_i \to A_k$ bedeutet, dass Job A_i fertig bearbeitet sein muß, bevor Job A_k startet.
$\beta_2 = \circ$	Es gibt keine Vorrangbedingungen.
$\beta_2 = chain$	Die Vorrangbedingungen haben Wegstruktur.
$\beta_2 = outtree$	Jeder Job hat höchstens einen Vorgänger.
$\beta_2 = intree$	Jeder Job hat höchstens einen Nachfolger.
$\beta_2 = tree$	Die Vorrangbedingungen haben Baumstruktur.
$\beta_2 = prec$	Die Vorrangbedingungen sind durch einen azyklischen Digraphen gegeben.
$\beta_3 \in \{\circ, r_i \geq 0\}$	Bereitstellungszeiten der Jobs
$\beta_3 = \circ$	Jeder Job ist zum Zeitpunkt 0 verfügbar: $r_i = 0 \quad \forall\, i \in I$.
$\beta_3 = r_i \geq 0$	Für jeden Job ist ein Bereitstellungszeitpunkt $r_i \geq 0$ gegeben.
$\beta_4 \in \{\circ, d_i\}$	Fälligkeitstermine für die Jobs
$\beta_4 = \circ$	Es gibt keine.
$\beta_4 = d_i$	Jeder Job muß vor seinem Fälligkeitstermin $d_i \geq 0$ fertig sein.
$\beta_5 \in \{\circ, p_{ij} = 1\}$	Spezielle Bearbeitungszeiten
$\beta_5 = \circ$	$p_{ij} \geq 0\ \forall\, (i,j) \in I \times J$, p_{ij}: natürliche Zahlen.
$\beta_5 = p_{ij} = 1$ für $\alpha_1 \in \{1, O, F, J\}$	$p_{ij} = 1$ gilt für alle Operationen $(ij) \in SIJ$.

Tabelle 2.3: Einige zusätzliche Bedingungen in β

Zielfunktionen $\gamma \in \{f_{max}, \sum f_i\}$	
$f_{max} \in \{C_{max}, L_{max}\}$	Zielfunktion: $f_{max} \to \min!$
$C_{max} = \max_{i \in I}\{C_i\} \to \min!$ $L_{max} = \max_{i \in I}\{L_i\} \to \min!$	Minimiere die Gesamtbearbeitungszeit! Minimiere die maximale Terminabweichung!
$\sum f_i \in \{\sum C_i, \sum T_i, \sum U_i,$ $\sum w_i C_i, \sum w_i T_i, \sum w_i U_i\}$	Zielfunktion: $\sum f_i \to \min!$
$\sum C_i \to \min!$	Minimiere die Summe der Fertigstellungszeiten aller Jobs!
$\sum T_i \to \min!$	Minimiere die Summe der Verspätungen aller Jobs!
$\sum U_i \to \min!$	Minimiere die Anzahl der verspäteten Jobs!
$\sum w_i C_i \to \min!$	Minimiere die Summe aller gewichteten Fertigstellungszeiten der Jobs !
$\sum w_i T_i \to \min!$	Minimiere die Summe der gewichteten Verspätungen aller Jobs!
$\sum w_i U_i \to \min!$	Minimiere die Summe der Gewichte für die verspäteten Jobs!

Tabelle 2.4: Reguläre Zielfunktionen

2.3 Modelle für Shopprobleme

Um die Ein- und Ausgabe von LiSA zu verstehen, werden die in LiSA benutzten Modelle kurz erläutert. LiSA ist vorwiegend für Shopprobleme konzipiert, also für flow-, job- und open-shop Probleme, wobei aber auch eine Reihe von Algorithmen für Einmaschinenprobleme enthalten sind. Es wird vorausgesetzt, dass zu einem festen Zeitpunkt jeder Job höchstens auf einer Maschine bearbeitet wird und jede Maschine höchstens einen Job bearbeitet.

2.3.1 Pläne und Zeitpläne (Sequences and Schedules)

Zur Einführung der Begriffe Plan und Schedule werden die folgenden Graphen definiert, wobei die Menge der Knoten gerade die Menge der Operationen SIJ ist:

- Der Graph der technologischen Reihenfolgen $G(MO)$ enthält alle Bögen, die den direkten Vorrangbedingungen der technologischen Reihenfolgen der Maschinen für die Aufträge entsprechen.

- Der Graph der organisatorischen Reihenfolgen $G(JO)$ enthält alle Bögen, die den direkten Vorrangbedingungen der organisatorischen Reihenfolgen der Aufträge auf den Maschinen entsprechen.

- Der Graph $G(MO, JO) = (SIJ, A)$ enthält alle Bögen aus $G(MO)$ und $G(JO)$, d.h.

$$((ij),(kl)) \in A \Longleftrightarrow \begin{cases} (i = k \ \wedge \text{ nach der Bearbeitung von Job } A_i \text{ auf} \\ \ M_j \text{ wird dieser Job auf } M_l \text{ bearbeitet}) \ \vee \\ (j = l \ \wedge \text{ nachdem die Maschine } M_j \text{ den Job } A_i \text{ bearbeitet hat,} \\ \ \text{bearbeitet diese Maschine den Job } A_k.) \end{cases}$$

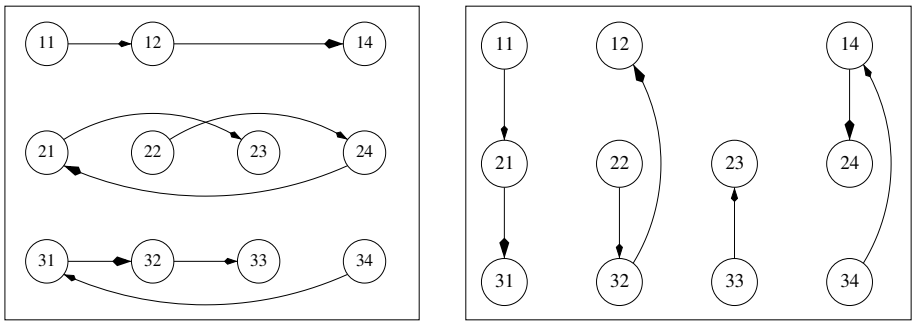
Eine Kombination von technologischen und organisatorischen Reihenfolgen wird *zulässig* genannt, wenn der zugehörende Graph $G(MO, JO)$ zyklenfrei ist. In diesem Fall heißt der Graph *Plangraph*.

Beispiel 1 *Drei Jobs sollen auf vier Maschinen bearbeitet werden. Die Bearbeitungszeitmatrix PT und die technologischen und organisatorischen Reihenfolgen sind gegeben durch*

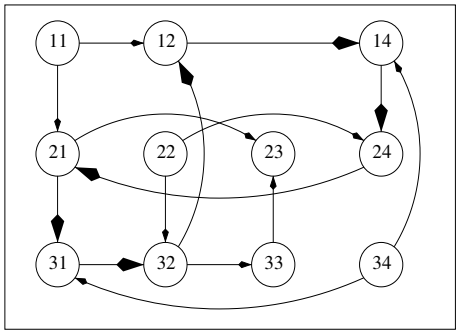
$$PT = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 3 & 4 & 3 \\ 1 & 5 & 1 & 2 \end{bmatrix}, \ \textit{deshalb gilt } SIJ = I \times J \backslash \{(13)\}.$$

$$\begin{array}{lll} A_1: & M_1 \to M_2 \to M_4 & \qquad M_1: \ A_1 \to A_2 \to A_3 \\ A_2: & M_2 \to M_4 \to M_1 \to M_3 & \qquad M_2: \ A_2 \to A_3 \to A_1 \\ A_3: & M_4 \to M_1 \to M_2 \to M_3 & \qquad M_3: \ A_3 \to A_2 \\ & & \qquad M_4: \ A_3 \to A_1 \to A_2 \end{array}$$

Die folgende Abbildung zeigt die Graphen $G(MO)$, $G(JO)$ und $G(MO, JO)$:



G(MO) G(JO)



G(MO,JO)

Abbildung 2.1: $G(MO)$, $G(JO)$ und $G(MO, JO)$

Diese Kombination von technologischen und organisatorischen Reihenfolgen ist nicht zulässig, da der Digraph $G(MO, JO)$ den Zyklus

$$(1,2) \to (1,4) \to (2,4) \to (2,1) \to (3,1) \to (3,2) \to (1,2)$$

enthält. Dann wäre jede Operation auf dem Zyklus Vorgänger und Nachfolger von sich selbst, was natürlich logisch nicht geht.
Wenn wir für die technologischen und organisatorischen Reihenfolgen die natürliche Reihenfolge der Maschinen bzw. der Jobs wählen, kann der Digraph $G(MO, JO)$ keine Zyklen enthalten, da alle horizontalen Bögen von links nach rechts und alle vertikalen Bögen von oben nach unten gerichtet sind. In diesem Fall ist der Digraph $G(MO, JO)$ ein Plangraph.

Jetzt wird jeder Knoten (ij) des Plangraphen $G(MO, JO)$ mit der Bearbeitungszeit p_{ij} gewichtet. Dann bezeichnet man einen Zeitplan der Bearbeitung aller Operationen als *Schedule*. Schedules werden gewöhnlich durch die Start- oder Endzeitpunkte der Bearbeitung aller Operationen beschrieben und durch *Ganttdiagramme* visualisiert, die *maschinenorientiert* oder *auftragsorientiert* sein können. Es existieren die folgenden Klassen von Schedules:

Ein Schedule heißt *semiaktiv*, wenn keine Operation eher fertig werden kann, ohne dass man den zugehörenden Plan ändert.

Ein Schedule heißt *aktiv*, wenn keine Operation früher fertig werden kann, ohne dass irgendeine andere Operation später fertig wird.

Ein Schedule heißt *non-delay*, wenn keine Maschine stillsteht, solange es einen verfügbaren Job gibt, der auf dieser Maschine zu bearbeiten ist.

Jeder non-delay Schedule ist aktiv und jeder aktive Schedule ist semiaktiv. Die Umkehrung ist im allgemeinen nicht gültig.

Beispiel 2 *Zu der Bearbeitungsmatrix aus Beispiel 1 ist der folgende azyklische Digraph $G(MO, JO)$ gegeben. Damit kann der zugehörende semiaktive Schedule in einem auftragsorientierten Ganttdiagramm visualisiert werden. Zunächst werden alle Operationen bearbeitet, die Quellen (Knoten ohne Vorgänger) im Plangraphen sind. Dann streicht man die Quellen und alle von ihnen ausgehenden Bögen und ordnet die Operationen als nächste an, die nun Quellen sind, usw. Der Schedule ist darüberhinaus aktiv, aber im open-shop Fall nicht non-delay, da die Operation* (11) *zum Zeitpunkt 2 starten könnte.*

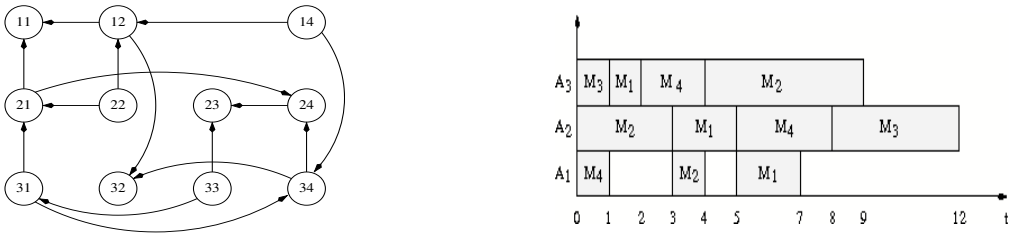


Abbildung 2.2: Plangraph $G(MO, JO)$ und Schedule (auftragsorientiertes Ganttdiagramm)

2.3.2 Das Block-Matrizen Modell für Shopprobleme

Für ein Shopproblem lassen sich die technologischen und organisatorischen Reihenfolgen, die Pläne und Schedules durch Matrizen beschreiben. Jede der Matrizen enthält in der i-ten Zeile und j-ten Spalte eine Information zu der Operation (ij). Dies ist entweder eine strukturelle oder zeitliche Eigenschaft der Operation (ij). Dazu wird der Rang $rg(v)$ eines Knotens v in einem azyklischen Digraphen eingeführt. Bezeichnet man die Anzahl der Knoten auf einem Weg w in einem azyklischen Digraphen als Länge des Weges, so ist der Rang $rg(v)$ die Knotenanzahl eines längsten Weges, der im Knoten v endet. Werden die Knoten (ij) des Plangraphen $G(MO, JO)$ mit der Bearbeitungszeit p_{ij} gewichtet, ergibt sich die Gesamtbearbeitungszeit des zugehörenden semiaktiven Schedules als Gewicht eines kritischen Weges (Weg mit maximalem Gewicht) in dem Plangraphen.

Damit ergeben sich zur Beschreibung der Graphen die folgenden Matrizen:

- $G(MO)$ wird durch $MO = [mo_{ij}]$ beschrieben, wobei mo_{ij} der Rang der Operation (ij) in $G(MO)$ ist (Matrix der technologischen Reihenfolgen).
- $G(JO)$ wird durch $JO = [jo_{ij}]$ beschrieben, wobei jo_{ij} der Rang der Operation (ij) in $G(JO)$ ist (Matrix der organisatorischen Reihenfolgen).
- $G(MO, JO)$ wird durch $LR = [lr_{ij}]$ beschrieben, wobei lr_{ij} der Rang der Operation (ij) im Plangraphen $G(MO, JO)$ ist (Plan).

In jeder Zeile i der Matrix MO steht eine Permutation der Zahlen $1. \ldots, u_i$, wobei u_i die Anzahl der Operationen des Auftrags A_i ist. In jeder Spalte j der Matrix JO steht eine Permutation der Zahlen $1, \ldots, v_j$, wobei v_j die Anzahl der Operationen ist, die auf der Maschine M_j bearbeitet werden sollen. Ein Plan LR vereint die Eigenschaften von MO und JO. Jeder Plan erfüllt aufgrund der Definition des Ranges die *Planeigenschaft*: Zu jeder Zahl $z = lr_{ij} > 1$ existiert in der Zeile i oder in der Spalte j oder in beiden die Zahl $z - 1$. Es sei bemerkt, dass Pläne über vollständigen Operationenmengen $SIJ = I \times J$ spezielle lateinische Rechtecke sind. Ein lateinisches Rechteck $LR[n, m, r]$ ist eine Matrix mit n-Zeilen, m-Spalten und Einträgen aus einer Belegungsmenge $\{1, \ldots, r\}$, wobei jede Zahl aus der Belegungsmenge in jeder Zeile und jeder Spalte höchstens einmal vorkommt. Erfüllt ein lateinisches Rechteck die Planbedingung, so ist es die Rangmatrix eines Plangraphen $G(MO, JO)$. Bei unvollständigen Operationenmengen entstehen sogenannte partielle lateinische Rechtecke als Rangmatrizen.

Zur Beschreibung eines semiaktiven Schedules, der eineindeutig zu einem Plan mit gegebener Matrix der Bearbeitungszeiten gehört, werden die folgenden Matrizen eingeführt:

- Die Matrix $C = [c_{ij}]$, wobei c_{ij} die Fertigstellungszeit für die Operationen (ij) ist.
- Die Matrix $H = [h_{ij}]$, wobei h_{ij} die minimale Zeit der Bearbeitung aller Vorgänger der Operationen (ij) ist (Head von (ij): frühester Starttermin der Operation (ij)).
- Die Matrix $T = [t_{ij}]$, wobei t_{ij} die minimale Zeit für die Bearbeitung aller Nachfolger der Operationen (ij) ist (Tail von (ij)).

- Die Matrix $W = H + PT + T = C + T = [w_{ij}]$, wobei w_{ij} das Gewicht eines schwersten Weges ist, der über die Operation (ij) führt.

Damit ergibt sich die Gesamtbearbeitungszeit

$$C_{max} = \max_{(ij) \in SIJ} c_{ij} = \max_{(ij) \in SIJ} w_{ij}.$$

Alle Operationen (ij) mit $w_{ij} = C_{max}$ liegen auf mindestens einem kritischen Weg. Mit Hilfe der Matrix W kann man durch Breitensuche alle kritischen Wege konstruieren. Die folgende Abbildung enthält das Block-Matrizen Modell:

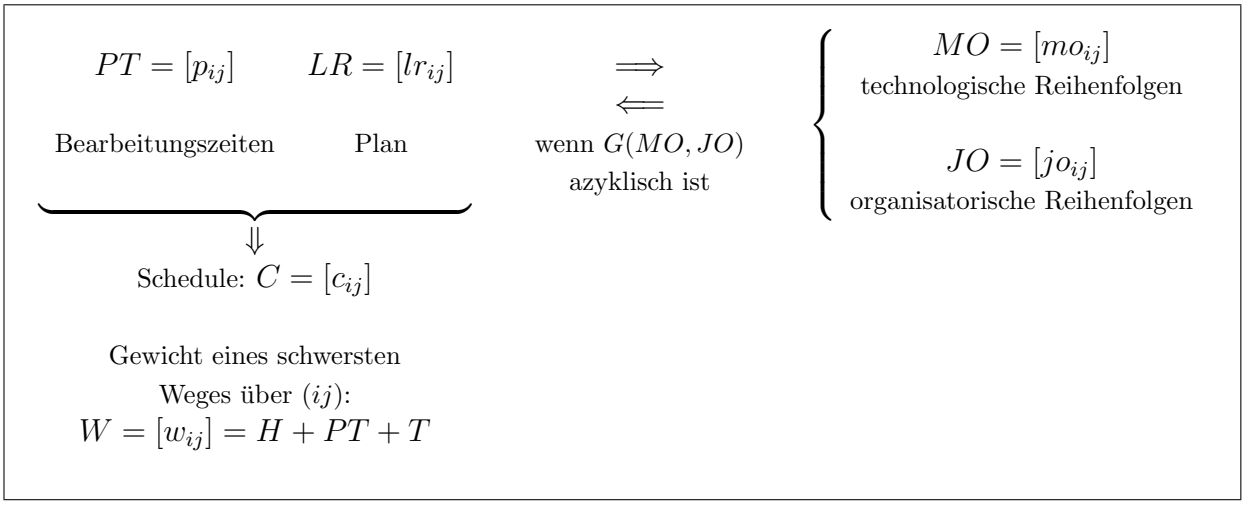


Abbildung 2.3: Block-Matrizen Modell

2.3.3 Disjunktives Graphenmodell und Block-Matrizen Modell

Shop-Probleme werden in der Literatur häufig durch das *disjunktive Graphenmodell* modelliert, vergleiche z. B. Brucker, [8]. Die Menge der Knoten des disjunktiven Graphen sind die Operationen, die gewöhnlich durchnummeriert werden und mit der Bearbeitungszeit gewichtet sind. Zwei Knoten sind durch eine Kante verbunden, wenn sie nicht gleichzeitig bearbeitet werden können, d.h. wenn sie zum gleichen Job oder zur gleichen Maschine gehören. Von einer fiktiven Quelle gibt es Bögen zu jedem Knoten. Von jedem Knoten gibt es Bögen zu einer fiktiven Senke. Vorrangbedingungen für die Operationen im job-shop und flow-shop Fall werden dann durch Richtung der entsprechenden Kante modelliert. Gesucht ist eine Orientierung der noch ungerichteten Kanten so, dass der Graph zyklenfrei ist und dass das Gewicht des kritischen Weges (bei Makespanminimierung) minimal wird.

Das in LiSA verwendete Modell kann von diesem durch die folgenden Überlegungen abgeleitet werden:
- Streiche die Quelle und die Senke und die mit ihnen inzidierenden Bögen aus dem disjunktiven Graphen.
- Bestimme eine azyklische Orientierung des disjunktiven Graphen.
Wenn man weiter alle bezüglich der Technologie und Organisation transitiven Bögen aus der azyklischen Orientierung des disjunktiven Graphen streicht, erhält man den in LiSA

benutzten Plangraphen $G(MO, JO)$. Also ist ein Plan eine Zusammenfassung aller Ränge einer azyklischen Orientierung eines disjunktiven Graphen.

2.3.4 Basisalgorithmen zum Block-Matrizen Modell

Zum Block-Matrizen Modell gehören einige Basisalgorithmen, die aufgrund der speziellen Struktur des Plangraphen meist linearen Aufwand haben. Sie benutzen ausnahmslos die Matrizen des Modells, die in eineindeutiger Weise die betrachteten graphentheoretischen Eigenschaften beschreiben.

Algorithmus 1 bestimmt den Plan LR aus gegebenen Matrizen MO und JO, falls der Graph $G(MO, JO)$ keine Zyklen enthält. Die Menge MQ enthält alle Operationen, die sowohl Quellen in $G(MO)$ als auch in $G(JO)$ sind.

Algorithmus 1: Berechnung von LR aus einer Kombination (MO,JO), wenn $G(MO, JO)$ keinen Zyklus enthält

---

Input: n, m, SIJ, MO und JO über der Operationenmenge SIJ;
Output: LR über der Operationenmenge SIJ, wenn $G(MO, JO)$ azyklisch ist;
BEGIN $k := 0$;
  REPEAT
    $k := k + 1$; Bestimme die Menge $MQ = \{(ij) \in SIJ \mid mo_{ij} = jo_{ij} = 1\}$;
    IF $MQ = \emptyset$ **THEN** (MO, JO) ist unzulässig und **STOP**;
    FORALL$(ij) \in MQ$ **DO**
      BEGIN
        $lr_{ij} = k$; Markiere in MO die Zeile i und in JO die Spalte j;
      END;
    $SIJ := SIJ \setminus MQ$;
    FORALL $(ij) \in SIJ$ in einer markierten Zeile in MO **DO** $mo_{ij} := mo_{ij} - 1$;
    FORALL $(ij) \in SIJ$ in einer markierten Spalte in JO **DO** $jo_{ij} := jo_{ij} - 1$;
  UNTIL $SIJ = \emptyset$;
END.

Algorithmus 2 berechnet MO und JO aus LR. Hier sind a_i and b_j die kleinsten natürlichen Zahlen, die für den Rang einer Operation (ij) verfügbar sind. Das maximale Element in LR wird mit r bezeichnet.

Algorithmus 2: Berechnung von MO und JO aus LR

Input: n, m, r, I, J, SIJ, LR über der Menge der Operationen SIJ;
Output: MO und JO über der Menge der Operationen SIJ;
BEGIN Setze $\forall\ i \in I$: $a_i = 1$ und $\forall\ j \in J$: $b_j = 1$;
  FOR $k := 1$ **TO** r **DO**
    FORALL $(ij) \in SIJ$ mit $lr_{ij} = k$ **DO**
      BEGIN
        Setze $mo_{ij} = a_i$ und $a_i = a_i + 1$;
        Setze $jo_{ij} = b_i$ und $b_j = b_j + 1$;
      END;
END.

Algorithmus 3 erzeugt einen semiaktiven Schedule, d.h. die Matrix $C = [c_{ij}]$ der Endbearbeitungszeitpunkte aller Operationen, aus der Matrix der Bearbeitungszeiten PT und einem Plan LR. Hier bezeichnen r_i und $\overline{r}_j$ jeweils den kleinsten möglichen Startzeitpunkt für Job A_i bzw. der Maschine M_j.

Algorithmus 3: Berechnung von $C.$ bei gegebenen PT und LR

Input: n, m, r, I, J, SIJ, PT und LR über der Operationenmenge SIJ;
Output: C über der Operationenmenge SIJ.
BEGIN
  Setze $\forall\ i \in I$: $r_i = 0$ und $\forall\ j \in J$: $\overline{r}_j = 0$;
  FOR $k := 1$ **TO** r **DO**
    FORALL $(ij) \in SIJ$ mit $lr_{ij} = k$ **DO**
      BEGIN
        $c_{ij} := \max\{r_i, \overline{r}_j\} + p_{ij}$;
        $r_i := c_{ij}$; $\overline{r}_j := c_{ij}$;
      END;
END.

Algorithmus 4 bestimmt die Matrizen $H = [h_{ij}]$ und $T = [t_{ij}]$. h_{ij} ist der Head der Operation (ij), d.h. der kleinsten Zeit, die für die Bearbeitung aller Vorgängeroperationen von (ij) im Plangraphen $G(MO, JO)$ nötig ist. t_{ij} bezeichnet den Tail der Operation (ij), d.h. der kleinsten Zeit, die für die Bearbeitung aller Nachfolgeoperationen von (ij) im Plangraphen $G(MO, JO)$ nötig ist. Hier sind $r_i, \overline{r}_j$ wieder die frühesten Startzeitpunkte des Jobs A_i bzw. der Maschine M_j. $s_i, \overline{s}_j$ bezeichnen die frühesten Startzeitpunkte des Jobs A_i bzw. der Maschine M_j bei der Rückwärtsrechnung.

Algorithmus 4: Berechnung von H und T

Input: n, m, r, I, J, SIJ, PT und LR über der Operationenmenge SIJ;
Output: H und T über der Operationenmenge SIJ.
BEGIN
Setze $\forall\ i \in I$: $r_i = 0$ und $\forall\ j \in J$: $\overline{r}_j = 0$;
Setze $\forall\ i \in I$: $s_i = 0$ und $\forall\ j \in J$: $\overline{s}_j = 0$;
FOR $k := 1$ **TO** r **DO**
BEGIN
FORALL $(ij) \in SIJ$ mit $lr_{ij} = k$ **DO**
BEGIN
$h_{ij} := \max\{r_i, \overline{r}_j\}$; $r_i := h_{ij} + p_{ij}$; $\overline{r}_j := h_{ij} + p_{ij}$;
END;
FORALL $(ij) \in SIJ$ mit $lr_{ij} = r - k + 1$ **DO**
BEGIN
$t_{ij} := \max\{s_i, \overline{s}_j\}$; $s_i := t_{ij} + p_{ij}$; $\overline{s}_j := t_{ij} + p_{ij}$;
END;
END;
END.

Die Matrix $W = H + PT + T$ enthält das Gewicht eines schwersten Weges w_{ij} von einer Quelle über die Operation (ij) zu einer Senke des Plangraphen $G(MO, JO)$. Also liegen alle Operationen mit maximalem Gewicht w_{ij} auf mindestens einem kritischen Weg. Wegen der Eigenschaften eines lateinischen Rechtecks lassen sich alle Operationen in $O(nm)$ nach nichtfallenden Rängen sortieren. Damit lassen sich die Heads und Tails sowie alle w_{ij} auch in linearer Zeit berechnen.
Das Kapitel schließt mit einem Beispiel zum Block-Matrizen Modell.

Beispiel 3 *Betrachtet wird die Matrix PT der Bearbeitungszeiten aus Beispiel 1. Die Duedates der Aufträge seien gegeben durch $d_1 = 6, d_2 = 12, d_3 = 8$. Die folgende Kombination aus technologischen und organisatorischen Reihenfolgen ist zulässig, weil der zugehörende Graph $G[MO, JO)$ keine Zyklen enthält, also ein Plangraph ist.*

$$
\begin{aligned}
A_1 &: \ M_4 \to M_2 \to M_1 \\
A_2 &: \ M_2 \to M_1 \to M_4 \to M_3 \\
A_3 &: \ M_3 \to M_1 \to M_4 \to M_2
\end{aligned}
$$

$$
\begin{aligned}
M_1 &: \ A_3 \to A_2 \to A_1 \\
M_2 &: \ A_2 \to A_1 \to A_3 \\
M_3 &: \ A_3 \to A_2 \\
M_4 &: \ A_1 \to A_3 \to A_2
\end{aligned}
$$

Abbildung 2.4: Technologische und organisatorische Reihenfolgen und der Plangraph $G(MO, JO)$

Algorithmus 1 erstellt den Plan LR und Algorithmus 3 berechnet den Schedule C:

$$\underbrace{PT = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 3 & 4 & 3 \\ 1 & 5 & 1 & 2 \end{bmatrix} \quad LR = \begin{bmatrix} 4 & 2 & - & 1 \\ 3 & 1 & 5 & 4 \\ 2 & 4 & 1 & 3 \end{bmatrix}}_{C = \begin{bmatrix} 7 & 4 & - & 1 \\ 5 & 3 & 12 & 8 \\ 2 & 9 & 1 & 4 \end{bmatrix}} \iff \begin{cases} MO = \begin{bmatrix} 3 & 2 & - & 1 \\ 2 & 1 & 4 & 3 \\ 2 & 4 & 1 & 3 \end{bmatrix} \\ JO = \begin{bmatrix} 3 & 2 & - & 1 \\ 2 & 1 & 2 & 3 \\ 1 & 3 & 1 & 2 \end{bmatrix} \end{cases}$$

Die Matrizen H und T werden mit Algorithmus 4 bestimmt, woraus sich die Matrix $W = H + PT + T$ ergibt:

$$W = \begin{bmatrix} 5 & 3 & - & 0 \\ 3 & 0 & 8 & 5 \\ 1 & 4 & 0 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 1 & - & 1 \\ 2 & 3 & 4 & 3 \\ 1 & 5 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 5 & - & 9 \\ 7 & 9 & 0 & 4 \\ 9 & 0 & 10 & 7 \end{bmatrix} = \begin{bmatrix} 7 & 9 & - & 10 \\ 12 & 12 & 12 & 12 \\ 11 & 9 & 11 & 11 \end{bmatrix}$$

Der Schedule C liefert $C_{max} = 12$ und $C_1 = 7$, $C_2 = 12$, $C_3 = 9$, womit $\sum C_i = 28$, $L_{max} = 1$, $\sum T_i = 2$ und $\sum U_i = 1$ folgt. Dieser Schedule ist optimal im open-shop Fall für die Zielfunktionen C_{max} and L_{max}, es gibt bessere Schedules für $\sum C_i$, $\sum T_i$ und $\sum U_i$.

In einem job-shop bzw. flow-shop Problem mit gegebener Matrix MO sind alle Pläne LR zulässig, die die gegebenen technologischen Reihenfolgen enthalten.

2.4 Algorithmenübersicht

In diesem Kapitel wird zunächst eine Übersicht zu den in LiSA verfügbaren Algorithmen gegeben. Ausführliche Beschreibungen sind im Kapitel 4 enthalten.

2.4.1 Universelle Algorithmen

- **Exakte Algorithmen**

Ein allgemeiner Branch and Bound Algorithmus, der auf der Insertion-Technik aus Bräsel [5] und Bräsel u.a. [7] basiert, löst die meisten open-, flow- und job-shop Probleme mit regulären Zielfunktionen. Der Nutzer kann hier untere und obere Schranken eingeben, wobei eine obere Schranke durch eine vorher angewendete Heuristik ermittelt werden kann. Dieses Verfahren ist nur für kleine Parameter n, m sinnvoll, da das Verfahren aufgrund seiner Universalität einen hohen Aufwand hat. Die Verwendung von guten unteren Schranken können die Laufzeit erheblich verringern.

- **Konstruktive Heuristiken**

Einfachen Reihungsregeln (Dispatching Rules) sind für eine große Reihe von Problemen verfügbar, auch unter zusätzlichen Bedingungen, wie z.B. Bereitstellungszeiten der Jobs. Schritt für Schritt wird eine neue Operation nach einer gegebenen Strategie angefügt.

In LiSA Version 3.0 sind für viele Probleme sogenannte Beam-Search Verfahren enthalten. Dies sind abgerüstete Branch-and-Bound Verfahren, d.h. der Branch-and Bound Baum wird nur teilweise erzeugt. Der Algorithmus bleibt polynomial, da eine Lösung durch Tiefensuche bestimmt wird und in jedem Schritt nur eine beschränkte Anzahl von Knoten (Beamweite)

Regel	Auswahl der nächsten Operation
RAND	zufällig (random)
FCFS	erste zur Verfügung stehende Operation (first come, first serve)
EDD	mit kleinstem Fälligkeitstermin (earliest due date first)
LQUE	mit kleinster Differenz von Fälligkeitstermin und (Bearbeitungszeit + Tail)
SPT	mit kleinster Bearbeitungszeit (shortest processing time first)
WSPT	mit kleinstem Quotienten aus Bearbeitungszeit und Gewicht (weighted shortest processing time first)
ECT	mit kleinster Fertigstellungszeit ((reachable) earliest completion time first)
WI	mit größtem Gewicht (largest weight first)
LPT	mit größter Bearbeitungszeit (largest processing time first)

Tabelle 2.5: Reihungsregeln

weiterverzweigt wird. Diese Verfahren unterteilen sich in Beam-Insert und Beam-Append Verfahren, je nachdem, ob die nächste Operation ein- oder angefügt wird.

- Iterative Algorithmen
In LiSA sind verschiedene Nachbarschaftssuchverfahren verfügbar. Jeder Knoten eines Nachbarschaftsgraphen entspricht einem Plan und wird mit dem Zielfunktionswert des jeweils betrachteten Problems gewichtet. Die Menge der Kanten ist in den verschiedenen Nachbarschaften unterschiedlich. Zum Start der Verfahren ist eine erste Lösung notwendig, die mit einem einfachen Konstruktionsverfahren ermittelt werden kann. Dann beginnt die iterative Suche auf dem Nachbarschaftsgraphen, um eine bessere Lösung zu finden.

Methode	Beschreibung
Iterative Improvement	Übergang zu einer besseren Lösung - nach Enumeration aller Nachbarn oder - wenn der erste bessere Nachbar gefunden wurde.
Simulated Annealing	erlaubt den Übergang zu einer schlechteren Lösung mit einer gewissen Wahrscheinlichkeit, die sich Schritt für Schritt verkleinert.
Threshold Accepting	erlaubt den Übergang zu einer schlechteren Lösung durch einen Schwellwert, der sich verkleinert.
Tabu Search	erzeugt eine Tabu-Liste, um die Rückkehr zu bereits betrachteten Lösungen zu verhindern.

Tabelle 2.6: Nachbarschafts-Suchstrategien

In Tabelle 2.6 sind die verschiedenen implementierten Suchmethoden erklärt. Um mehr Informationen zu erhalten werden, verweisen wir auf die Bücher von Brucker [8], Blazewicz

u.a. [3] oder Pinedo [21]. In LiSA sind eine Reihe von Nachbarschaften verfügbar, wie z.B. die API (adjacent pairwise interchange), bei der zwei benachbarte Operationen bzgl. Technologie oder Organisation miteinander vertauscht werden. In Kapitel 4.3 ist eine Übersicht aller in LiSA verfügbaren Nachbarschaften enthalten.
In LiSA-Version 3.0 sind auch genetische Algorithmen implementiert. Ausgehend von einer Population (Menge von Startlösungen) wird eine neue Generation erzeugt, wobei bestimmte Vererbungsgesetze (Mutation und Kreuzung) gelten. Das Maß der Güte einer Lösung ist ihr Zielfunktionswert. Wie in der Natur, setzen sich die besseren Lösungen im Laufe der Generationen durch.

2.4.2 Spezielle Algorithmen

Einige Algorithmen in LiSA sind speziellen Problemklassen gewidmet.
Tabelle 2.7 enthält exakte Algorithmen und Heuristiken. Die meisten der Algorithmen aus dieser Tabelle findet man in jedem Lehrbuch zur Schedulingtheorie. Die exakten Algorithmen zur Minimierung der Gesamtbearbeitungszeit im job und open-shop Fall vom Brucker-Team sind im Netz verfügbar und wurden in LiSA mit Erlaubnis eingebunden.

In LiSA - Version 3.0 gibt es keine Algorithmen für
- Probleme mit Vorrangbedingungen,
- Basisalgorithmen und Visualisierung von Problemen mit Unterbrechung der Operationen,
- Parallelmaschinenprobleme.

2.5 Das File Format in LiSA

LiSA benutzt zur Verarbeitung von Schedulingproblemen Dateien im XML-Format. In diesen Dateien – im Folgenden auch *Dokumente* genannt – werden Problemtypen in $\alpha \mid \beta \mid \gamma$ – Notation, Probleminstanzen und Schedules gespeichert. Darüber hinaus gibt es weitere Dokumente, die zur Algorithmenbeschreibung und zur Übergabe von Programmparametern an LiSA dienen.
Bei der Arbeit mit der LiSA-Oberfläche sind keine Kenntnisse über den Aufbau der Dokumente nötig. Dort treten sie nur über die Menüpunkte ***Datei/Speichern als*** und ***Datei/Oeffnen*** in Erscheinung, wo sie zur Abspeicherung und zur Anzeige bereits bearbeiteter Schedulingprobleme dienen. Beim externen Aufruf der LiSA-Algorithmen ($\rightarrow$ 7.1) oder der automatischen Abarbeitung von Algorithmen durch **auto_alg** ($\rightarrow$ 7.3) ist die Kenntnis über den genauen Aufbau dieser Dateien jedoch notwendig.
Die XML-Dokumente lassen sich, je nach ihrer Funktion, einem *Dokumenttyp* zuordnen. Im Einzelnen gibt es fünf Dokumenttypen:

- `problem` enthält nur einen Problemtyp in $\alpha \mid \beta \mid \gamma$ – Notation
- `instance` enthält neben dem Problemtyp auch eine Probleminstanz
- `schedule` enthält zusätzlich ein oder mehrere Pläne
- `algorithm` enthält eine Algorithmenbeschreibung
- `controls` enthält Programmparameter für LiSA

Problemtyp	Exakte Verfahren
$1 \mid\mid L_{max}$	Branch and Bound
F2$\mid\mid C_{max}$	Johnson Regel
J2$\mid\mid C_{max}$	Jackson regel
O2$\mid\mid C_{max}$	Gonzales/ Sahni Algorithmus
J$\mid\mid C_{max}$	Originaler Brucker's Branch-and-Bound Algorithmus
O$\mid\mid C_{max}$	Originaler Brucker's Branch-and-Bound Algorithmus
O $\mid$pmtn $\mid C_{max}$	Gonzales/ Sahni (ohne Visualisierung)
O2$\mid\mid \ C_{max}$	LAPT Regel, Job mit längster Bearbeitung auf der anderen Maschine zuerst (longest alternating processing time first)
Problemtyp	**Heuristiken**
$1 \mid\mid \sum w_i T_i$	Smith Regel (arbeitet wie die WSPT Regel)
$1 \mid\mid L_{max}$	ERD Regel, Job mit kleinster Bereitstellungszeit zuerst (earliest release date first)
F $\mid\mid C_{max}$	Beam Insert mit Beamweite 1 auf der Menge der permutation-flow-shop Pläne
J $\mid\mid C_{max}$	Shifting bottleneck Heuristik [1]
O $\mid\mid C_{max}$ O $\mid\mid \sum C_i$	Matching Heuristiken [7]

Tabelle 2.7: Algorithmen für spezielle Probleme

Alle XML-Dateien in LiSA beginnen mit den folgenden Zeilen:

```
<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE instance PUBLIC "" "LiSA.dtd">
```

Die erste Zeile ist der typische Dateikopf für XML-Dateien und muss für jede neue Datei übernommen werden. Sie beschreibt die verwendete XML-Version und die Zeichencodierung (hier latin-1). Die zweite Zeile gibt den Dokumenttyp (hier `instance`) an. Zusätzlich wird hier eine Datei LiSA.dtd angegeben. Sie enthält eine Strukturbeschreibung (eine sogenannte *document type definititon*, DTD), die dem XML-Interpreter hilft, den korrekten Aufbau des Dokumentes zu überprüfen. Diese Datei befindet sich je einmal in den Unterordnern `bin` und `data` des LiSA-Hauptverzeichnisses.

2.5.1 Der Dokumenttyp `problem`

Eine XML-Datei vom Dokumenttyp `problem` stellt nur eine Problembeschreibung in der üblichen $\alpha \mid \beta \mid \gamma$ – Notation dar. Hier eine Beispieldatei mit der Codierung des Problemtyps $F \mid r_i, intree \mid C_{max}$:

```
<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE problem PUBLIC "" "LiSA.dtd">
<problem xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
    <alpha env="F" />
    <beta release_times="yes" prec="intree" />
    <gamma objective="Sum_Ci" />
</problem>
```

Eine vollständige Liste für die Codierung aller Problemtypen in XML findet sich im Anhang A.

2.5.2 Der Dokumenttyp `instance`

Der Dokumenttyp `instance` enthält eine Probleminstanz, also neben dem Problemtyp auch Parameter wie Problemgröße, Bearbeitungszeitenmatrix und Fälligkeitstermine. Diese Parameter werden zusammengefasst in einem `<values>`-Element, das dem bereits vorgestellten `<problem>`-Element folgt. Die Problemgröße, die Bearbeitungszeitenmatrix und die Matrix der Operationenmenge müssen immer angegeben werden; alle weiteren Parameter sind optional.

Ein Beispiel mit dem Problemtyp $O \mid\mid \sum T_i$ der Größe n=5, m=10 mit gegebenen Bearbeitungszeiten, Fälligkeitsterminen und Operationenmenge:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE instance PUBLIC "" "LiSA.dtd">
<instance xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
  <problem>
    <alpha env="O"/>
    <beta/>
    <gamma objective="Sum_Ti"/>
```

```
  </problem>
  <values m="10" n="5">
    <processing_times model="lisa_native">
      {
       {  28  32  93  71  42  18  24  10   1   0 }
       {  65  15  68  88  39   0  15  84  39  59 }
       {  15  98  56  59  56  95   0  49  15  25 }
       {  65  80  63  32  62  96  13  20  13  46 }
       {  15  39  43  94  21  25  41  48   3  90 }
      }
    </processing_times>
    <operation_set model="lisa_native">
      {
       {   1   1   1   1   1   1   1   1   1   0 }
       {   1   1   1   1   1   0   1   1   1   1 }
       {   1   1   1   1   1   1   0   1   1   1 }
       {   1   1   1   1   1   1   1   1   1   1 }
       {   1   1   1   1   1   1   1   1   1   1 }
      }
    </operation_set>
    <due_dates>
      {  95  94  39  27  69 }
    </due_dates>
  </values>
</instance>
```

Im Anhang A finden sich Codierungen für weitere Parameter, die zur Probleminstanz gehören.

Dieser Dokumenttyp wird von den Algorithmen als Eingabe akzeptiert, d.h. einem Algorithmus kann durch einen **externen Aufruf** (→ 7.1) ein solches Dokument übergeben werden. Dieser berechnet für die in dem Dokument enthaltene Probleminstanz eine oder auch mehrere Lösungen.
Zusätzlich zu den Parametern, die eine Probleminstanz ausmachen, lassen sich in `instance`-Dokumenten auch Aufrufparameter für einen Lösungsalgorithmus ablegen. Diese Aufrufparameter steuern das Verhalten eines Algorithmus, dem das betreffende Dokument als Eingabe übergeben werden soll und gehören nicht zur eigentlichen Probleminstanz. Sie werden in einem `<controls>`-Element zusammengefasst, das dem `<values>`-Element folgt. Der Inhalt dieses `<controls>`-Elementes ist für jeden Lösungsalgorithmus individuell. Näheres hierzu wird im Abschnitt **Algorithmenmodule** (→ 7.1) behandelt. In Kapitel 4 sind die Aufrufparameter der Algorithmen dokumentiert.
Probleminstanzen können leicht über die LiSA-Oberfläche erzeugt werden. Nach der Angabe eines Problemtyps über ***Datei/Neu*** und der Eingabe der Parameter mit ***Bearbeiten/Parameter*** kann man diese Daten mit ***Datei/Speichern als*** in Form eines `instance`-Dokumentes abspeichern.

2.5.3 Der Dokumenttyp `solution`

Ein `solution`-Dokument ist ähnlich aufgebaut wie ein `instance`-Dokument, mit dem Unterschied, dass zusätzlich ein oder mehrere Lösungen in Form von Plänen enthalten sind.

Daneben können auch weitere Daten wie Fertigstellungszeiten oder technologische und organisatorische Reihenfolgen gespeichert werden. Alle Daten, die zu einer Lösung gehören, werden in einem `<schedule>`-Element zusammengefasst.

Ein Beispiel mit einer Lösung, die als Plan und Matrix der Fertigstellungszeiten vorliegt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE solution PUBLIC "" "LiSA.dtd">
<solution xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
  <problem>
    <alpha env="0"/>
    <beta/>
    <gamma objective="Sum_Ui"/>
  </problem>
  <values m="5" n="3">
    <processing_times model="lisa_native">
      {
       {  38  25   9  48  23 }
       {  59  25  65  88  82 }
       {   9  48  57  10   5 }
      }
    </processing_times>
    <operation_set>
      {
       { 1  1  1  1  1 }
       { 1  1  1  1  1 }
       { 1  1  1  1  1 }
      }
    </operation_set>
    <due_dates>
      { 107 175  71 }
    </due_dates>
  </values>
  <schedule m="5" n="3" semiactive="yes">
    <plan model="lisa_native">
      {
       {   4   3   1   5   2 }
       {   5   4   2   7   6 }
       {   2   5   6   3   1 }
      }
    </plan>
    <completion_times model="lisa_native">
      {
       {  95  57   9 143  32 }
       { 158  99  74 328 240 }
       {  14 147 204  24   5 }
      }
    </completion_times>
  </schedule>
</solution>
```

Dokumente vom Typ `solution` werden von den Lösungsalgorithmen als Ausgabe erzeugt. Sie können aber auch, wie `instance`-Dokumente, als Eingabe für einen Algorithmus dienen. In diesem Fall werden allerdings die bereits enthaltenen Lösungen verworfen. Wie auch ein `instance`-Dokument muss ein `solution`-Dokument beim manuellen Algorithmenaufruf ein `<controls>`-Element zur Parameterübergabe enthalten. Dieses wird zwischen dem `<values>`- und dem `<schedule>`-Element eingeschoben.

2.5.4 Der Dokumenttyp `algorithm`

Dieser Dokumenttyp ist für das Einfügen neuer Algorithmen in LiSA interessant. Sämtliche Algorithmen werden durch solche Dokumente in das Programm eingebunden. Dazu durchsucht LiSA die Unterordner `data/alg_desc/language/english` bzw. `data/alg_desc/language/german` (je nach eingestellter Sprache) beim Programmstart nach `algorithm`-Dokumenten und bindet die in ihnen beschriebenen Algorithmen dann in das Menü ***Algorithmen*** ein.
Damit ein Algorithmus richtig auf eine Probleminstanz angewendet werden kann, müssen in diesem Dokument verschiedene Informationen angegeben werden. Dazu gehört unter Anderem der Problemtyp, den der Algorithmus behandeln kann, oder Aufrufparameter, die übergeben werden müssen.
Auf ein Beispiel soll an dieser Stelle verzichtet werden, da diese Dokumente für die eigentliche Arbeit mit LiSA uninteressant sind. Ein Beispieldokument findet sich jedoch im Unterordner `src/algorithm/sample` (sofern das LiSA-Quellcodepaket installiert ist). Näheres zu `algorithm`-Dokumenten findet sich auch in den Abschitten **Algorithmenmodule** (→ 7.1) und **Einbinden externer Algorithmen** (→ 7.2).

2.5.5 Der Dokumenttyp `controls`

In einem Dokument vom Typ `controls` werden die Programmoptionen von LiSA gespeichert. Für jeden Benutzer gibt es ein solches Dokument, das unter `~/.lisa/default.xml` zu finden ist. Der genaue Aufbau und die änderbaren Optionen werden im Anhang A beschrieben. Die wichtigsten der hier gespeicherten Optionen lassen sich auch im Menüpunkt **Optionen/allgemeine Einstellungen** eingeben, so dass das manuelle Bearbeiten dieses Dokumentes selten nötig ist.

Kapitel 3

Eingabe

Eine Probleminstanz eines in LiSA zu lösenden Problems besteht aus dem **Problemtyp** (→ 3.1), der Anzahl der Jobs n, der Anzahl der Maschinen m und den **Parametern** (→ 3.2), die durch den Problemtyp sowie n und m erforderlich sind. Die Eingabe kann per Hand oder durch das Öffnen einer Eingabedatei erfolgen. In LiSA gibt es einen Zufallszahlengenerator, der gleichverteilte ganzzahlige Daten aus einem einzugebenden Intervall erzeugt. Zum Starten des Generators sind die beiden ganzzahligen zufälligen Werte Time Seed und Machine Seed notwendig. die gewöhnlich aus der Systemzeit generiert werden, aber auch durch Handeingabe eingestellt werden können. Dies erlaubt einen Wiederaufruf von generierten Problemen.

3.1 Problemtyp

Um ein Scheduling-Problem mit LiSA behandeln zu können, muss zuerst der Problemtyp in der 3-Feld-Notation $\alpha \mid \beta \mid \gamma$ nach Graham u.a. [15]. festgelegt werden. Daneben wird die Maschinen- und die Jobanzahl eingegeben. Dabei gilt ein Problemtyp als zulässig, wenn mindestens die Maschinenumgebung und eine Zielfunktion gewählt wurde. Wurde zusätzlich eine Maschinen- und Auftragsanzahl angegeben, kann später das Parameterfenster zur Eingabe und Veränderung der problemspezifischen Daten angewählt werden.

Aufruf

Nach Aufruf des Menüpunktes ***Datei/Neu*** zur Eingabe eines neuen Problemtyps oder ***Bearbeiten/Problemtyp*** zur Editierung eines bereits vorhandenen Problemtyps öffnet sich das Problemtyp-Fenster.

Einstellungen

Die Problemtypeingabe folgt den in jedem Scheduling-Lehrbuch üblichen Konventionen, vergleiche auch Kapitel **Definitionen und Bezeichnungen** 2.1, deshalb wird an dieser Stelle auf eine ausführliche Beschreibung aller Möglichkeiten für die Maschinenumgebung, für die Nebenbedingungen und für die Zielfunktion verzichtet.

- **Maschinenumgebung**
- **Nebenbedingungen**
- **Zielfunktion**

- **(Anzahl der) Maschinen**
- **(Anzahl der) Aufträge**

Irreguläre Zielfunktionen

Zusätzlich können folgende Zielfunktionen angegeben werden:
$IRREG1 = \Sigma \mid C_i - d_i \mid$
$IRREG2 = w_{late} L_{max} + w_{early} \max(d_i - C_i)^+ + \Sigma w_i T_i + \Sigma w_{early}(d_i - C_i)^+$
Man beachte, dass die in dieser Zielfunktion vorkommenden zusätzlichen Gewichte von einem externen Problemgenerator erzeugt werden müssen.

Bemerkungen

LiSA versucht stets, die letzte Eingabe umzusetzen. Wenn man eine inkonsistente Eingabe macht, modifiziert LiSA gegebenenfalls frühere Eingaben. Gibt man zum Beispiel für ein Einmaschinenproblem Mehrmaschinenaufträge an, so wird der Problemtyp automatisch auf P geändert.
Da die Klassifikation ständig erweitert wird, kann es sein, dass bestimmte Felder noch nicht implementiert sind. LiSA erlaubt schon jetzt die Eingabe von mehreren zehntausend verschiedenen Problemtypen.
Nicht zu jedem auswählbaren Problemtyp ist schon ein Lösungsverfahren implementiert. Passende Algorithmen sind in den Untermenüs von ***Algorithmen/Exakte Verfahren*** und ***Algorithmen/Heuristische Verfahren*** auszuwählen, sofern LiSA auch schon die Parameter kennt.

3.2 Parameter

Wenn LiSA den Problemtyp, die Anzahl der Maschinen und die Anzahl der Aufträge kennt, kann die Eingabe sämtlicher Parameter erfolgen.

Aufruf

Nach Aufruf des Menüpunktes ***Bearbeiten/Parameter*** öffnet sich das Fenster **Parameter** zur Eingabe der Parameter, das je nach Problemtyp sowie n und m verschieden aufgebaut ist. Die folgenden Auswahlmöglichkeiten entstehen für ein Problem $J \mid r_i \mid \Sigma w_i T_i$.

Einstellungen

Ansicht: Die Bereitstellungszeiten r_i, die Fälligkeitstermine d_i und die Gewichte w_i werden bei folgenden Auswahlmöglichkeiten jedesmal mitangezeigt:

- ***Operationenmenge:*** Der Eintrag in Zeile i und Spalte j dieser Matrix ist 1, wenn die Operation (ij) existiert und 0 im anderen Fall.
- ***Technologische Reihenfolge:*** Der Eintrag in Zeile i und Spalte j dieser Matrix ist k, wenn die Maschine M_j an Position k in der technologischen Reihenfolge der Maschinen für Auftrag J_i steht.
- ***Bearbeitungszeiten:*** Der Eintrag in Zeile i und Spalte j dieser Matrix ist die Bearbeitungszeit der Operation (ij).

Erzeugen: Will man die Daten per Hand eingeben, muss man zunächst eine Ansicht aufrufen, die die einzugebenden Daten enthält. Dann läßt sich durch Anklicken eines Feldes ein Wert für dieses Feld eingeben. Der Kursor springt dann auf das nächste Feld, usw.. Es folgt je nach Problemtyp die Eingabe

- **... der Bearbeitungszeiten**
- **... der Operationenmenge**
- **... der Technologien**
- **... der Gewichte**
- **... der Fälligkeitstermine**
- **... der Bereitstellungszeiten**

Durch Wahl von **Erzeugen** öffnet sich das Fenster **Erzeugen** zum zufälligen Generieren von gleichverteilten Daten. Gewählt werden können

- **Minimum** ist die kleinste Zahl des Zufallszahlenintervalls.
- **Maximum** ist die größte Zahl des Zufallszahlenintervalls.
- **Time Seed** ist eine Zufallszahl zur Erzeugung der numerischen Daten.
- **Machine Seed** ist eine Zufallszahl zur Erzeugung von Reihenfolgen der Operationen auf den Maschinen.

Durch die Parameter Time Seed und Machine Seed ist die Wiederholbarkeit der Erzeugung einer Probleminstanz gewährleistet, weiterführende Informationen zu diesem Zufallszahlengenerator in Tailard [24].

Technologien uebernehmen: Die technologischen Reihenfolgen werden nur dann übernommen, wenn in jeder Zeile i eine Permutation der Zahlen 1 bis Anzahl der Operationen für Auftrag J_i steht.

ACHTUNG!

Man kann den Menüpunkt ***Bearbeiten/Parameter*** auch zum Editieren der Probleminstanz des gewählten Problemtyps benutzen. LiSA versucht stets, die letzte Eingabe umzusetzen. Wird eine inkonsistente Eingabe gemacht, modifiziert LiSA gegebenenfalls frühere Eingaben, so dass hier bei mehrfacher Anwendung des Editierens Fehler entstehen können.

3.3 Eingabedatei

Den Problemtyp und alle Werte einer Probleminstanz kann LiSA auch aus einer Eingabedatei in XML-Format entnehmen. Jede von LiSA selbst abgespeicherte Datei kann ebenfalls wieder geöffnet werden, hier ist schon ein Schedule mitenthalten, dessen Gantt-Diagramm standardmässig angezeigt wird.

Nähere Informationen zum Dateiformat: **Das File Format in LiSA**→ 2.5

LiSA ist in der Lage, eine Liste von Plänen aus einer xml-Datei zu verarbeiten, die als Ergebnis eines auto-alg Aufrufs (siehe **Automatisierter Algorithmenaufruf** → 7.3) entsteht. Nach dem Aufruf einer solchen XML-Datei findet man die Liste im Menüpunkt ***Bearbeiten/Liste von Plaenen***. In dem sich öffnenden Fenster **List of Sequences** sind die in der Liste enthaltenen Pläne in einer Tabelle zu sehen, sie sind in umgekehrter Reihenfolge ihres Aufrufs in der auto_alg Datei angeordnet. Gleichzeitig werden die Zielfunktionswerte und weitere selbsterklärende Eigenschaften angezeigt. Die Liste läßt sich nach einer Zielfunktion oder nach einer der Eigenschaften sortieren.

Durch Anklicken eines Feldes (ungleich 0) kann man sich die Informationen des Plans, die im Kapitel 5 **Ausgabe** beschrieben werden, auf den Bildschirm holen.

ACHTUNG!

Bei Weiterverarbeitung eines Plans aus einer Liste geht die Liste verloren.

Kapitel 4

Algorithmen in LiSA

4.1 Universelle exakte Verfahren

Als exakter Algorithmus zum Lösen von Schedulingproblemen steht in LiSA im Wesentlichen ein universeller Branch & Bound Algorithmus zur Verfügung.
Ein Branch & Bound Algorithmus unterscheidet sich von einer vollständigen Enumeration aller Lösungen dadurch, dass Teile des Lösungsbaumes ausgeblendet werden. Um sicherzustellen, dass dabei keine optimalen Lösungen mitausgeblendet werden, wird jede generierte Lösung mit einer Schranke bewertet, die für den gesamten an dieser Lösung hängenden Ast des Lösungsbaumes gilt. Ist nun die Bewertung einer erzeugten Lösung schlechter als die beste bereits bekannte Lösung, so kann der gesamte Ast zu dieser Lösung ignoriert werden.
Neben einem **allgemeinen Branch & Bound Algorithmus** (→ 4.1.1), der für jeden Problemtyp und jede Zielfunktion anwendbar ist, stehen für die $C_{\max}$-Zielfunktion auch spezielle Open-Shop und Job-Shop Branch & Bound Algorithmen nach **Brucker** (→ 4.4.1) Verfügung.

4.1.1 Universeller Branch & Bound Algorithmus

Mit diesem Branch & Bound Algorithmus können exakte Lösungen für alle open-shop, flow-shop, job-shop, oder Einmaschinenprobleme mit regulärer Zielfunktion ermittelt werden. Sollte für den gewählten Problemtyp ein alternatives exaktes Verfahren verfügbar sein, so sollte dieses gewählt werden, da die Rechenzeit für den universellen Branch & Bound Algorithmus unter Umständen beträchtlich sein kann.
Bei geeigneten Einschränkungen (siehe Einstellungen) kann dieser Algorithmus auch zur Bestimmung einer guten Näherungslösung verwendet werden.

Aufruf

Nach der Eingabe von **Problemtyp** und den erforderlichen **Parametern** kann dieses Verfahren im Menüpunkt ***Exakte Verfahren/Branch & Bound*** im Menü ***Algorithmen*** erreicht werden.

Einstellungen

Anzahl der Lösungen: Hier kann eine obere Schranke für die Anzahl der auszugebenden Pläne angegeben werden. Die Standardeinstellung beträgt 1. Der Algorithmus ist in der Lage, *alle* optimalen Pläne zu berechnen. Für diesen Fall muss in diesem Feld eine Zahl eingetragen werden, die mindestens so groß wie die tatsächliche Anzahl aller optimalen Pläne ist.

Untere Schranke: Hier kann eine bekannte untere Schranke angegeben werden. LiSA behandelt jeden Plan, dessen Zielfunktionswert diesen Wert nicht überschreitet, als optimal. Wird hier ein Wert eingegeben, der größer als der optimale Zielfunktionswert ist, so kann das zur Ausgabe von nicht optimalen Plänen führen. Auf diese Weise kann hier auch ein Level der Zielfunktion angegeben werden, der als Näherung akzeptiert wird.

Obere Schranke: Hier kann eine bekannte obere Schranke für den Zielfunktionswert einer optimalen Lösung eingegeben werden. Eine solche Schranke ergibt sich beispielsweise durch den Zielfunktionswert einer bekannten (Näherungs-) Lösung für das vorliegende Problem. LiSA verwirft dann alle Teilpläne mit größerem Zielfunktionswert.

Einfügereihenfolge: Durch die Auswahl der Einfügereihenfolge wird festgelegt, in welcher Reihenfolge die Operationen in den Teilplänen angefügt werden.

- **LPT** (longest processing time): Die Operationen werden abhängig von der Größe der Bearbeitungszeiten angefügt, beginnend mit der längsten Bearbeitungszeit.
- **RANDOM:** Die Operationen werden in zufälliger Reihenfolge angefügt.

Art der unteren Schranke: Hiermit wird festgelegt, mit welcher Prozedur LiSA untere Schranken für den Zielfunktionswert berechnet.

- **NORMAL:** Als untere Schranke wird der Zielfunktionswert des Teilplans benutzt.
- **EXTENDED:** (Diese erweiterte Methode ist noch nicht implementiert.)

Problembehandlung

Dieser Branch & Bound Algorithmus ist ein Universalsolver und deshalb sehr aufwendig. Sofern möglich, sollte ein problemspezifisches Verfahren benutzt werden. Gestaltet sich die Laufzeit zu lang, so kann der Algorithmus jederzeit abgebrochen werden. Dieser Vorgang kann jedoch selbst einige Minuten dauern. Man erhält dann eine Näherungslösung als Ausgabe.
Wenn mehrere optimale Lösungen gefunden werden sollen, so empfiehlt es sich, zunächst den optimalen Zielfunktionswert mit der Einstellung 1 für die **Anzahl der Lösungen** zu bestimmen, und anschließend weitere Optimallösungen in einem zweiten Lauf mit nun bekannten Schranken zu berechnen.

Aufruf in der Autoalg-Eingabedatei

In einer Eingabedatei für den automatisiereten Algorithmenaufruf (→ 7.3) wird der allgemeine Branch & Bound Algorithmus folgendermaßen aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE bb
  string INS_ORDER RANDOM
  long NB_SOLUTIONS 1000000
  double UPPER_BOUND 10000000
  double LOWER_BOUND -10000000
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für das Branch & Bound Verfahren bb aufgerufen werden. Optional können dann die **Einfügereihenfolge**, die **Anzahl der Lösungen** sowie jeweils eine **obere** bzw. eine **untere Schranke** angegeben werden.

4.2 Universelle konstruktive Verfahren

Heuristische Verfahren dienen zur schnellen Bestimmung einer Näherungslösung. Im Gegensatz zu **exakten Verfahren** (→ 4.1) liefern sie Lösungen mit deutlich geringerer Rechenzeit. Allerdings können diese Lösungen unter Umständen weit vom Optimum entfernt sein.
In LiSA sind neben einfachen **konstruktiven Heuristiken (Reihungsregeln)** (→ 4.2.1), **Matchingheuristiken** (→ 4.2.2) auch Beam-Search Verfahren mit **Anfügen** (→ 4.2.3) bzw. **Einfügen** (→ 4.2.4) von Operationen enthalten.

4.2.1 Reihungsregeln

Reihungsregeln (Dispatching Rules) sind sehr schnelle Heuristiken, bei denen Schritt für Schritt neue Operationen angefügt werden. Die jeweils anzufügenden Operationen werden dabei durch die gewählte Strategie bestimmt.
Einige Problemklassen können mit Hilfe solcher einfachen Verfahren sogar exakt gelöst werden.

Aufruf

Nach Eingabe von ***Problemtyp*** und erforderlichen ***Parametern*** wählt man ***Heuristische Verfahren/Reihungsregeln*** im Menü ***Algorithmen***. Abhängig vom Problemtyp können einzelne Vorrangregeln auch unter ***Exakte Verfahren*** aufgelistet sein. In diesem Fall können meist keine Einstellungen getroffen werden.

Einstellungen

Erzeuge Schedule:

- **SEMIAKTIV:** Ein Schedule heißt semiaktiv, wenn keine Operation früher begonnen werden kann, ohne den Plan zu verändern, d.h. in einem semiaktiven Schedule treten keine unnötigen Stillstandszeiten auf.
- **AKTIV:** Ein Schedule heißt aktiv, wenn er semiaktiv ist, und keine Stillstandszeit so existiert, dass eine spätere Operation auf der betreffenden Maschine in dieser Stillstandszeit bearbeitet werden könnte.
- **NON-DELAY:** Ein Schedule heißt non-delay, wenn zu jedem Zeitpunkt, an dem eine Maschine M_j und ein Auftrag A_i zur Verfügung stehen, mit der Bearbeitung von Operation (ij) begonnen wird. Ein non-delay-Schedule ist dann auch aktiv.

Prioritätsregel: Es stehen folgende Prioritäten zur Auswahl. Die einzufügenden Operationen werden dann gemäß der gewählten Strategie geordnet und dieser Reihenfolge an den Plan angefügt.

- **RAND** (random): Die Operationen werden zufällig angeordnet.

- **FCFS** (first come - first serve): Es wird die Operation als nächste angeordnet, die schon am längsten bereitsteht.
- **EDD** (earliest due date): Die Operationen werden aufsteigend nach ihren Fälligkeitsterminen angeordnet.
- **LQUE**: Es wird die Operation als nächste angeordnet, die die kleinste Differenz 'Fälligkeitstermin − (Bearbeitungszeit + Tail)' hat. Im Fall von open-shop wird statt des Tails die Summe der Bearbeitungszeiten aller noch nicht angeordneten Operationen des jeweiligen Auftrags betrachtet. Als zweitrangige Auswahlregel wird die Operation gewählt, auf deren Maschine die Summe der Bearbeitungszeiten der noch nicht angeordneten Operationen am größten ist.
- **SPT** (shortest processing time): Die Operationen werden gemäß ihrer Bearbeitungszeiten angeordnet, beginnend mit der kleinsten.
- **WSPT** (weighted shortest processing time): Es wird die Operation mit dem kleinsten Quotienten aus Bearbeitungszeit und Gewicht als nächste angeordnet.
- **ECT** (earliest completion time): Es wird die Operation als nächste angeordnet, die am schnellsten fertiggestellt werden kann.
- **WI**: Die Aufträge sind gewichtet, und die Operationen werden gemäß der Gewichte der Aufträge angeordnet, beginnend mit dem größten Gewicht.
- **LPT** (longest processing time): Die Operationen werden gemäß ihrer Bearbeitungszeiten angeordnet, beginnend mit der größten.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden Reihungsregeln wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE dispatch
  string SCHEDULE ACTIVE
  string RULE SPT
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Reihungsregeln dispatch aufgerufen werden. Optional sind die Anforderung, was für ein Plan **erzeugt** werden soll, und welche **Reihungsregel** angewandt werden soll.

4.2.2 Matchingheuristiken

Diese Heuristiken erzeugen einen Plan für open-shop Probleme. Dabei wird das gegebene Problem auf Zuordnungsprobleme zurückgeführt, die optimal gelöst werden. Allen Operationen einer optimalen Zuordnung wird der gleiche Rang zugewiesen, d.h. es entstehen stets rangminimale Pläne.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Matching Heuristiken*** im Menü ***Algorithmen***.

Einstellungen

Art des Algorithmus: Mit dieser Option wird festgelegt, wie mit den durch die **Art des Matchings** festgelegten Gewichten der Operationen umgegangen wird.

- **BOTTLENECK:** Die Operationen werden nach ihren Gewichten sortiert. Anschließend wird das größte Gewicht p bestimmt, so dass $M = \{pt_{ij} \mid pt_{ij} \geq p\}$ ein perfektes Matching enthält. Die entsprechenden Operationen aus diesem Matching werden an den Plan angefügt und die Gewichte gelöscht. Dieser Vorgang wird solange wiederholt, bis alle Operationen an den Plan angefügt sind.
- **WEIGHTED:** Es wird ein maximal gewichtetes Matching berechnet. Die entsprechenden Operationen werden an den Plan angefügt und die Gewichte gelöscht (d.h. auf den kleinsten zulässigen Wert gesetzt). Dieser Vorgang wird solange wiederholt, bis alle Operationen an den Plan angefügt sind. (Dies ist die Standardeinstellung.)

Art des Matchings: Mit dieser Option wird festgelegt, wie die Gewichte der einzelnen Operationen berechnet werden.

- **HEADS:** Die erste anzufügende Menge wird mit dem MIN Parameter berechnet. Nachdem eine Anfangsmenge von Operationen angefügt wurde, werden für alle verbleibenden Operationen die Heads bestimmt. Diese werden zur Bearbeitungszeit addiert und das Ergebnis wird vom größten möglichen Wert abgezogen. Dies ergibt die Gewichte für das nächste Matching. Auf diese Weise minimiert der Algorithmus in jedem Schritt die aktuelle Gesamtbearbeitungszeit.
- **MIN:** Das Gewicht einer Operation ergibt sich aus der Differenz zwischen der Bearbeitungszeit der Operation und dem größten zulässigen Wert. Auf diese Weise wird die durchschnittliche Bearbeitungszeit aller Operationen in einem Anfügeschritt minimiert.
- **MAX:** Das Gewicht einer Operation ergibt sich direkt aus der Bearbeitungszeit. Auf diese Weise wird die durchschnittliche Bearbeitungszeit aller Operationen in einem Anfügeschritt maximiert. (Dies ist die Standardeinstellung.)

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden Matching Heuristiken wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE match
  string MINMAX MAX
  string TYPEOF WEIGHTED
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für die Matching Heuristiken match aufgerufen werden. Optional können dann die **Art des Matchings** und die **Art des Algorithmus** angegeben werden.

4.2.3 Beam Search Verfahren mit Anfügetechnik

Beam Search Verfahren sind eine heuristische Variante des **Branch & Bound Verfahren** (→ 4.1.1). Im Unterschied zu diesem ist bei ihnen die Anzahl der untersuchten Teillösungen in jedem Knoten des Lösungsbaumes beschränkt. Das führt zu einer unvollständigen Suche im Lösungsraum, die möglicherweise sämtliche optimalen Lösungen ausschließt. Allerdings sind Verfahren dieser Art jedoch auch bedeutend schneller als ein umfangreicher Branch & Bound Algorithmus. Die Qualität der Lösung hängt dabei sowohl vom gewählten Problemtyp, als auch von den gewählten Einstellungen ab.
Beim **Anfüge**-Verfahren werden für jeden Teilplan verschiedene Positionen betrachtet, in die neue Operationen mit aufsteigendem Rang angefügt werden können. Dabei wird mit unterschiedlichen Strategien jeweils nach einer Operation gesucht, die möglichst gut an diese Stelle passt. Neben dem Anfüge-Verfahren ist auch ein **Einfüge**-Verfahren (→ 4.2.4) in LiSA implementiert.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Beam Search (Anfügen)*** im Menü ***Algorithmen***.

Einstellungen

Anzahl zu verfolgender Zweige: Hier wird festgelegt, wie viele der in jedem Schritt neu erzeugten Teilpläne weiterbetrachtet werden sollen.

Auswahlmethode: Hier wird festgelegt, wie die gewählte Anzahl der Teilpläne, die weiter betrachtet werden sollen (**Anzahl zu verfolgender Zweige**), ausgewählt werden. Die Teilpläne werden dabei mit ihren Zielfunktionswerten bewertet.

- **INSERT1:** Aus der Gesamtheit *aller* in einem Iterationsschritt zur Verfügung stehenden Kinder-Pläne werden die besten ausgewählt.
- **INSERT2:** Es wird für jeden Vater-Plan genau ein (bester) Kind-Plan ausgewählt, unabhängig davon, ob dieser global erfolgversprechend ist oder nicht.

Auswahlkriterium: Diese Option steht nur bei $C_{\max}$-Problemen zur Verfügung. Sie stellt dort eine alternative Bewertungsmethode für die Teilpläne zur Verfügung.

- **CLAST:** Als Bewertung eines Teilplans dienen die Kosten eines längsten Weges durch die Operation, die als letzte eingefügt wurde.
- **LB(Sum_Ci):** Als Bewertung eines Teilplans dient der Sum_Ci-Zielfunktionswert.
- **OBJECTIVE:** Der Zielfunktionswert wird zur Bewertung eines Teilplans benutzt. Dies ist die Standardmethode.

Tie Breaking: Ist die Auswahl der anzufügenden Operation nicht eindeutig, so legt diese Option fest, mit welcher Regel die gesuchte Operation unter den in Frage kommenden ausgewählt wird.

- **FCFS** (first come - first serve): Es wird die Operation als nächste angeordnet, die schon am längsten bereitsteht.
- **LPT** (longest processing time): Es wird die Operation mit der größten Bearbeitungszeit ausgewählt.
- **SPT** (shortest processing time): Es wird die Operation mit der kleinsten Bearbeitungszeit ausgewählt.
- **RANDOM:** Die Operation wird zufällig ausgewählt.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden Beam Search Anfüge-Verfahren wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE beam
  string MODE ATTACH
  string ATTACH_WHAT Machines+Jobs
  string BREAK_TIES FCFS
  string INS_METHOD INSERT1
  string CRITERION OBJECTIVE
  long K_BRANCHES 5
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Beam Search Verfahren beam aufgerufen werden. Die Angabe des Beam Search-Modus ist obligatorisch. Danach können optional die Parameter für das **Anfügekriterium** (ATTACH_WHAT mit Optionen Machines, Jobs, Machines+Jobs), das **Tie Breaking**, die **Auswahlmethode**, das **Auswahlkriterium**, und die **Anzahl zu verfolgender Zweige** gesetzt werden.

4.2.4 Beam Search Verfahren mit Einfügetechnik

Beam Search Verfahren sind eine heuristische Variante des **Branch & Bound Verfahrens** (→ 4.1.1). Im Unterschied zu diesem ist bei ihnen die Anzahl der untersuchten Teillösungen in jedem Knoten des Lösungsbaumes beschränkt. Das führt zu einer unvollständigen Suche im Lösungsraum, die möglicherweise sämtliche optimalen Lösungen ausschließt. Allerdings sind Verfahren dieser Art jedoch auch bedeutend schneller als ein umfangreicher Branch & Bound Algorithmus. Die Qualität der Lösung hängt dabei sowohl vom gewählten Problemtyp, als auch von den gewählten Einstellungen ab.
Beim **Einfüge**-Verfahren wird die Reihenfolge, in der die Operationen zu den jeweiligen Teilplänen hinzugefügt werden, vorher festgelegt. Für jeden Teilplan und jede neue Operation werden dann verschiedene zulässige Positionen betrachtet, in die die neue Operation mit einem von mehreren möglichen Rängen eingefügt werden kann. Neben dem Einfüge-Verfahren ist auch ein **Anfüge**-Verfahren (→ 4.2.3) in LiSA implementiert.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Beam Search (Einfügen)*** im Menü ***Algorithmen***.

Einstellungen

Anzahl zu verfolgender Zweige: Hier wird festgelegt, wie viele der in jedem Schritt neu erzeugten Teilpläne weiterbetrachtet werden sollen.

Einfügereihenfolge: Hier wird die Reihenfolge festgelegt, in der die Operationen zu den Teilplänen hinzugefügt werden sollen.

- **MACHINEWISE:** Die Operationen werden spaltenweise angeordnet.
- **DIAGONAL:** Diese Strategie basiert auf der Betrachtung von Diagonalen in einem Quadrat der Größe $M = \min(m, n)$.
- **QUEEN_SWEEP:** Diese Strategie basiert auf der Betrachtung von unabhängigen Mengen als Lösungen eines M-Damen Problems auf einem $M \times M$-Schachbrett ($M = \min(m, n)$).
- **RANDOM:** Die Operationen werden in zufälliger Reihenfolge eingefügt.
- **LPT** (longest processing time): Die Operationen werden nach fallenden Bearbeitungszeiten geordnet.
- **ECT** (earliest completion time): Die jeweils einzufügende Operation wird für jeden Teilplan so ausgewählt, dass die Fertigstellungszeit für den neuen Teilplan minimal wird. Im Unterschied zu allen anderen Strategien zur Auswahl der Einfügereihenfolge, ist diese Auswahl nicht global. In jedem Schritt wird für jeden betrachteten Teilplan die Operation bestimmt, die als nächste eingefügt wird.
- **SPT** (shortest processing time): Die Operationen werden nach aufsteigenden Bearbeitungszeiten geordnet.

Auswahlmethode: Hier wird festgelegt, wie die gewählte Anzahl der Teilpläne, die weiter betrachtet werden sollen (**Anzahl zu verfolgender Zweige**), ausgewählt werden. Die Teilpläne werden dabei mit ihren Zielfunktionswerten bewertet.

- **INSERT1:** Aus der Gesamtheit *aller* in einem Iterationsschritt zur Verfügung stehenden Kinder-Pläne werden die besten ausgewählt.
- **INSERT2:** Es wird für jeden Vater-Plan genau ein (bester) Kind-Plan ausgewählt, unabhängig davon, ob dieser global erfolgversprechend ist oder nicht.

Auswahlkriterium: Diese Option steht nur bei $C_{\max}$-Problemen zur Verfügung. Sie stellt eine alternative Bewertungsmethode für die Teilpläne zur Verfügung.

- **OBJECTIVE:** Der Zielfunktionswert wird zur Bewertung eines Teilplans benutzt. Dies ist die Standardmethode.
- **CLAST:** Als Bewertung eines Teilplans dienen die Kosten eines längsten Weges durch die Operation, die als letzte eingefügt wurde.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden Beam Search Einfüge-Verfahren wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE beam
  string MODE INSERT
  string INS_ORDER LPT
  string INS_METHOD INSERT1
  string CRITERION OBJECTIVE
  long k_BRANCHES 5
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Beam Search Verfahren beam aufgerufen werden. Die Angabe des Beam Search-Modus ist ebenso obligatorisch, wie eine Angabe zur **Einfügereihenfolge**. Danach können optional die Parameter für die **Auswahlmethode**, das **Auswahlkriterium** und die **Anzahl zu verfolgender Zweige** gesetzt werden.

4.3 Universelle iterative Verbesserungsverfahren

Iterative Verbesserungsverfahren versuchen, eine vorgegebene Startlösung, die z.B. durch eine schnelle Heuristik gefunden wurde, schrittweise zu verbessern. Dazu werden verschiedene Nachbarschaften zwischen Lösungen betrachtet, die mit unterschiedlichen Strategien abgesucht werden können. Anwendung finden solche Verfahren bei der Suche nach sehr guten Näherungslösungen in mittlerer Laufzeit.
Neben der einfachen **Iterativen Suche** (→ 4.3.1) und der **Tabu-Suche** (→ 4.3.2) sind in LiSA auch erweiterte Nachbarschaftssuchverfahren, wie **Simmulated Annealing** (→ 4.3.3) oder **Threshold Accepting** (→ 4.3.4) implementiert. Darüber hinaus stehen mit den **Genetischen Algorithmen** (→ 4.3.5) und einem **Ameisensuchverfahren** (→ 4.3.6) auch noch weitere sogenannte Metaheuristiken zur Verfügung.
Die Qualität der Lösung hängt dabei sowohl von dem verwendeten Verfahren, der Qualität der Startlösung, als auch von dem investierten Aufwand ab.

Nachbarschaften

In allen Verbesserungsverfahren spielen die verschiedenen Nachbarschaften eine besondere Rolle. Ein Nachbarschaftsgraph gibt Auskunft darüber, welche anderen Pläne von einem gegebenem Plan in einem Iterationsschritt erzeugt werden können. Die Wahl der Nachbarschaft kann großen Einfluss auf die Güte des Verfahrens haben. In LiSA sind die folgenden Nachbarschaften berücksichtigt, dabei steht nicht jede Nachbarschaft für jeden Problemtyp zur Verfügung.

- **k_API** (*k*-adjacent pairwise interchange)**:** *k + 1* benachbarte Operationen werden zufällig neu angeordnet.
- **SHIFT:** Eine Operation wird im Plan verschoben.

- **PI** (pairwise interchange)**:** Zwei beliebige Operationen werden vertauscht.
- **TRANS** (transpose)**:** Auf einer Maschine wird eine Teilfolge der auf ihr bearbeiteten Operationen umgedreht.
- **CR_API** (critical API)**:** Eine $C_{\max}$-kritische Operation wird mit einer direkt benachbarten Operation vertauscht.
- **SC_API** (semicritical API)**:** Eine $C_{\max}$-kritische Operation oder mit einer gewissen Wahrscheinlichkeit auch eine andere wird mit einer direkt benachbarten Operation vertauscht.
- **BL_API** (block API)**:** Eine $C_{\max}$-kritische Block-End-Operation wird mit einer direkt benachbarten Operation vertauscht.
- **CR_SHIFT** (critical SHIFT)**:** Eine $C_{\max}$-kritische Operation wird im Plan verschoben.
- **BL_SHIFT** (block SHIFT)**:** Eine $C_{\max}$-kritische Block-End-Operation wird im Plan verschoben.
- **CR_TRANS** (crtitical TRANS) **:** Auf einer Maschine wird die Reihenfolge der Jobs zwischen zwei kritischen Operationen umgedreht.
- **SC_TRANS** (semicrtitical TRANS) **:** Die Reihenfolge auf einer Maschine wird nicht immer zwischen zwei kritischen Operationen umgedreht, sondern mit einer kleinen Wahrscheinlichkeit auch zwischen nichtkritischen Operationen.
- **3_CR:** Eine $C_{\max}$-kritische Operation wird mit einer direkt benachbarten Operation vertauscht. Zusätzlich werden jeweils noch Vorgänger und Nachfolger mit anderen Operationen getauscht.
- **k_REINSERTION:** k beliebige Operationen werden aus dem Plan entfernt und anschließend wieder neu eingefügt.
- **CR_TRANS_MIX:** In 75% aller Fälle wird ein Nachbarplan aus der CR_TRANS-Nachbarschaft erstellt, ansonsten aus der CR_API-Nachbarschaft.
- **CR_SHIFT_MIX:** In 75% aller Fälle wird ein Nachbarplan aus der CR_SHIFT-Nachbarschaft erstellt, ansonsten aus der CR_API-Nachbarschaft.

4.3.1 Iterative Suche

Bei der Iterativen Suche wird in jedem Schritt die Nachbarschaft abgesucht und die erste bessere Lösung akzeptiert. Man endet dabei immer in einem lokalen Optimum, dessen Qualität jedoch unter Umständen sehr schlecht sein kann.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Iterative Suche*** im Menü ***Algorithmen***.

Einstellungen

Anzahl der Lösungen: Hiermit wird die maximale Anzahl von erzeugten Lösungen festgelegt.

Abbruch nach Stillständen (Abbruchkriterium): Hiermit kann ein Abbruchkriterium in Form einer maximalen Anzahl von Iterationsschritten ohne Verbesserung der bisher besten Lösung festgelegt werden. Ist dieser Wert größer als die **Anzahl der Lösungen**, so ist er unwirksam.

k für k-API oder k-REINSERTION: Hier wird der Parameter k angegeben, falls als **Nachbarschaft** k_API oder k_REINSERTION gewählt wird.

Abbruchschranke (Abbruchkriterium): Hier kann eine obere Schranke für die Akzeptanz eines Zielfunktionswertes angegeben werden. Die Nachbarschaftssuche wird abgebrochen, sobald der angegebene Zielfunktionswert erreicht oder unterschritten wurde.

Nachbarschaft: Hier wird die Definition der Nachbarschaft festgelegt, d.h. auf welche Weise aus einem Plan neue Pläne erzeugt werden.

- **k_API**
- **SHIFT**
- **PI**
- **CR_API**
- **BL_AP**
- **CR_SHIFT**
- **BL_SHIFT**
- **3_CR**
- **k_REINSERTION.**

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die Iterative Suche wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE nb_iter
  string METHOD IterativeImprovement
  string TYPE RAND
  string NGBH k_API
  long k 1
  long STEPS 5000
  long NUMB_STUCKS 214748000
  double ABORT_BOUND -214748000
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Nachbarschaftssuchverfahren Verfahren nb_iter aufgerufen werden. Für die Iterative Suche muss als Methode IterativeImprovement aufgerufen werden. Angaben zur Auswahl der **Nachbarschaft** sind obligatorisch (ggf. zusätzlich mit entsprechendem Parameter für k). Danach können optional die folgenden Parameter gesetzt werden: die **Gesamtzahl der erzeugten Pläne**, die **Anzahl der Stillstände**, nach denen abgebrochen wird, und eine etwaige **Abbruchschranke**.

4.3.2 Tabu-Suche

Bei der Tabu-Suche handelt es sich um eine einfache Erweiterung der Iterativen Suche. Um nicht zu schnell in einem schlechten lokalen Optimum stecken zu bleiben, wird immer der Nachbar mit dem besten Zielfunktionswert als nächste Lösung akzeptiert. Um bei einer Verschlechterung in einem Schritt zu verhindern, dass man im nächsten Schritt wieder zu der bereits betrachteten (besseren) Lösung zurückgeht, werden alle besuchten Lösungen in einer Tabu-Liste gespeichert und für eine nochmalige Betrachtung gesperrt. Um die Rechenzeit für die Vergleiche jeder Lösung mit dieser Tabu-Liste in Grenzen zu halten, wird die Liste in ihrer Länge begrenzt.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Tabu-Suche*** im Menü ***Algorithmen***.

Einstellungen

Anzahl der Lösungen: Hiermit wird die maximale Anzahl von erzeugten Lösungen festgelegt.

Tabulistenlänge: Dieser Parameter bestimmt, wie viele vergangene Lösungen im 'Gedächtnis' des Verfahrens bleiben und im aktuellen Schritt nicht erneut gewählt werden dürfen.

Anzahl der Nachbarn: Hier wird festgelegt, wie viele Nachbarn in jedem Schritt generiert werden sollen.

k für k-API oder k-REINSERTION: Hier wird der Parameter k angegeben, falls als **Nachbarschaft** k_API oder k_REINSERTION gewählt wird.

Abbruchschranke (Abbruchkriterium): Hier kann eine obere Schranke für die Akzeptanz eines Zielfunktionswertes angegeben werden. Die Nachbarschaftssuche wird abgebrochen, sobald der angegebene Zielfunktionswert erreicht oder unterschritten wurde.

Nachbarschaft: Hier wird die Definition der Nachbarschaft festgelegt, d.h. auf welche Weise aus einem Plan neue Pläne erzeugt werden.

- **k_API**
- **SHIFT**
- **PI**

- **CR_API**
- **BL_API**
- **CR_SHIFT**
- **BL_SHIFT**
- **3_CR**
- **k_REINSERTION.**

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die Tabu-Suche wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE nb_iter
  string METHOD TabuSearch
  string TYPE RAND
  string NGBH k_API
  long k 1
  long STEPS 1
  long TABULENGTH 1
  long NUMB_NGHB 50
  double ABORT_BOUND -214748000
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Nachbarschaftssuchverfahren Verfahren nb_iter aufgerufen werden. Für die Tabu Suche muss als Methode TabuSearch aufgerufen werden. Angaben zur Auswahl der **Nachbarschaft** sind obligatorisch (ggf. zusätzlich mit entsprechendem Parameter für k). Danach können optional die folgenden Parameter gesetzt werden: die **Gesamtzahl der erzeugten Pläne**, die **Länge der Tabuliste**, die **Anzahl der erzeugten Nachbarn**, eine etwaige **Abbruchschranke**.

4.3.3 Simulated Annealing

Simulated Annealing ist eine erweiterte Variante der Iterativen Suche. Hierbei darf die Verbesserung in jedem Schritt mit einer gewissen, kleiner werdenden Wahrscheinlichkeit auch negativ sein. Dadurch kann vermieden werden, dass man zu schnell in einem schlechten lokalen Optimum stecken bleibt. Die Wahrscheinlichkeit für die Akzeptanz einer schlechteren Lösung hängt von der 'Temperatur' ab, die im Laufe des Prozesses abkühlt. Der gesamte Prozess kann aus mehreren Abkühlungsperioden oder -zyklen bestehen, wobei die Temperatur zu Beginn jeder Periode erneut auf den Ausgangswert gesetzt wird.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Simulated Annealing*** im Menü ***Algorithmen***.

Einstellungen

Epochenlänge: Hier kann die Länge einer sogenannten Epoche festgesetzt werden. Während einer Epoche bleibt die Temperatur gleich. Erst zu Beginn der folgenden Epoche wird sie auf die nächste Stufe abgekühlt.

Anzahl der Nachbarn: Hiermit wird die Anzahl der Nachbarn der aktuellen Lösung festgelegt, die in jedem Iterationsschritt erzeugt werden sollen.

Anzahl der Iterationsschritte: Hiermit wird die maximale Anzahl der Iterationsschritte festgelegt.

Abbruch nach Stillständen (Abbruchkriterium): Hiermit kann ein Abbruchkriterium in Form einer maximalen Anzahl von Iterationsschritten ohne Verbesserung der bisher besten Lösung festgelegt werden. Ist dieser Wert größer als die **Anzahl der Lösungen**, so ist er unwirksam.

k für k-API oder k-REINSERTION: Hier wird der Parameter k angegeben, falls als **Nachbarschaft** k_API oder k_REINSERTION gewählt wird.

Starttemperatur: Dieser Parameter legt die Starttemperatur für den Abkühlungsprozess fest. Wird im Laufe der Abkühlungsperiode die **Endtemperatur** erreicht, so wird die Temperatur zurückgesetzt und eine neue Periode beginnt.

Endtemperatur: Dieser Parameter legt die Endtemperatur für eine Abkühlungsperiode fest. In Abhängigkeit vom **Abkühlungsparameter**, der **Epochenlänge** und der **Anzahl der Lösungen** kann diese Temperatur deutlich vor der Anzahl der Lösungen oder auch deutlich danach erreicht werden. Daraus ergeben sich im ersten Fall mehrere Abkühlungsperioden und im zweiten Fall eine unvollständige Periode.

Abkühlungsparameter: Dieser Parameter steuert die Geschwindigkeit der Abkühlung. Abhängig vom gewählten **Abkühlungsschema** bestimmt er, wie sich die Temperatur für die nächste Epoche aus der aktuellen Temperatur berechnet.

Abbruchschranke (Abbruchkriterium): Hier kann eine obere Schranke für die Akzeptanz eines Zielfunktionswertes angegeben werden. Die Nachbarschaftssuche wird abgebrochen, sobald der angegebene Zielfunktionswert erreicht oder unterschritten wurde.

Abkühlungsschema: Die Abkühlung der Temperatur von einer Epoche zur nächsten kann nach verschiedenen fallenden Funktionen bestimmt werden. Für jedes Schema kann der **Abkühlungsparameter** p $(0 < p < 1)$ jeweils explizit angegeben werden.

- **LINEAR**: Die neue Temperatur wird wie folgt aus der alten berechnet: $T^{new} = T^{old} - p_L \cdot T^{start}$.
- **GEOMETRIC**: Die neue Temperatur wird wie folgt aus der alten berechnet: $T^{new} = p_G \cdot T^{old}$.

- **LUNDYANDMEES**: Die neue Temperatur wird wie folgt aus der alten berechnet (nach Lundy-Mees): $T^{new} = T^{old}/(1 + p_{LM} \cdot T^{old})$.

Nachbarschaft: Hier wird die Definition der Nachbarschaft festgelegt, d.h. auf welche Weise aus einem Plan neue Pläne erzeugt werden.

- **k_API**
- **SHIFT**
- **PI**
- **CR_API**
- **BL_AP**
- **CR_SHIFT**
- **BL_SHIFT**
- **3_CR**
- **k_REINSERTION.**

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird Simulated Annealing wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE nb_iter
  string METHOD SimulatedAnnealingNew
  string NGBH k_API
  long k 1
  long EPOCH 100
  long NUMB_NGHB 1
  long STEPS 1
  long NUMB_STUCKS 214748000
  double TSTART 20
  double TEND 0.9
  double COOLPARAM 0.0005
  double ABORT_BOUND -214748000
  string COOLSCHEME LUNDYANDMEES
  string TYPE ENUM
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Nachbarschaftssuchverfahren Verfahren nb_iter aufgerufen werden. Für Simulated Annealing muss als Methode SimulatedAnnealingNew aufgerufen werden. Angaben zur Auswahl der **Nachbarschaft** sind obligatorisch (ggf. zusätzlich mit entsprechendem Parameter für k). Danach können optional die folgenden Parameter gesetzt werden: die **Anzahl der Epochen**, die **Anzahl der Nachbarn**, die **Anzahl der Iterationsschritte**, die **Anzahl der Stillstände**, nach denen abgebrochen wird, die

Starttemperatur, die **Endtemperatur**, den **Abkühlungsparameter**, eine etwaige **Abbruchschranke**, das **Abkühlungsschema** und die Art und Weise zur **Erzeugung der Nachbarn** (zufällig oder enumeriert) (Variable TYPE mit Optionen ENUM und RAND).

4.3.4 Threshold Accepting

Threshold Accepting ist ebenfalls eine erweiterte Version der Iterativen Suche. Im Unterschied zur Lokalen Suche werden in jedem Schritt auch kleine Verschlechterungen akzeptiert. Die Schwelle für die Akzeptanz schlechterer Lösungen geht jedoch im Laufe des Prozesses gegen Null. In LiSA ist eine lineare Verkleinerung der Threshold-Schranke implementiert.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Threshold Accepting*** im Menü ***Algorithmen***.

Einstellungen

Anzahl der Lösungen: Hiermit wird die maximale Anzahl von erzeugten Lösungen festgelegt.

Abbruch nach Stillständen (Abbruchkriterium): Hiermit kann ein Abbruchkriterium in Form einer maximalen Anzahl von Iterationsschritten ohne Verbesserung der bisher besten Lösung festgelegt werden. Ist dieser Wert größer als die **Anzahl der Lösungen**, so ist er unwirksam.

Anfangswahrscheinlichkeit: Dieser Parameter gibt die maximal zulässige Verschlechterung in Promille wieder.

Maximale Stillstandsanzahl: Um bei zu gering gewordener Wahrscheinlichkeit ein Stehenbleiben des Verfahrens in einem lokalen Optimum zu verhindern, wird nach der hier angegebenen Anzahl von Iterationen ohne Verbesserung der aktuellen Lösung die Wahrscheinlichkeit wieder auf ihren Anfangswert zurückgesetzt.

k für k-API oder k-REINSERTION: Hier wird der Parameter k angegeben, falls als **Nachbarschaft** k_API oder k_REINSERTION gewählt wird.

Abbruchschranke (Abbruchkriterium): Hier kann eine obere Schranke für die Akzeptanz eines Zielfunktionswertes angegeben werden. Die Nachbarschaftssuche wird abgebrochen, sobald der angegebene Zielfunktionswert erreicht oder unterschritten wurde.

Nachbarschaft: Hier wird die Definition der Nachbarschaft festgelegt, d.h. auf welche Weise aus einem Plan neue Pläne erzeugt werden.

- **k_API**
- **SHIFT**
- **PI**
- **CR_API**
- **BL_API**
- **CR_SHIFT**

- **BL_SHIFT**
- **3_CR**
- **k_REINSERTION.**

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird Threshold Accepting wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE nb_iter
  string METHOD ThresholdAccepting
  string TYPE RAND
  string NGBH k_API
  long k 1
  long STEPS 1
  long NUMB_STUCKS 214748000
  long PROB 1
  double ABORT_BOUND -214748000
  long MAX_STUCK 30
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle Nachbarschaftssuchverfahren Verfahren nb_iter aufgerufen werden. Für Threshold Accepting muss als Methode ThresholdAccepting aufgerufen werden. Angaben zur Auswahl der **Nachbarschaft** sind obligatorisch (ggf. zusätzlich mit entsprechendem Parameter für k). Danach können optional die folgenden Parameter gesetzt werden: die **Gesamtzahl der erzeugten Pläne**, die **Anzahl der Stillstände**, nach denen abgebrochen wird, die **Startwahrscheinlichkeit**, eine etwaige **Abbruchschranke**.

4.3.5 Genetische Algorithmen

Bei genetischen Algorithmen handelt es sich um Metaheuristiken, die aus einer Population von Startlösungen durch zufällige Änderungen neue Lösungen erzeugen. Dabei werden die biologischen Konzepte der Mutation und der Kreuzung zweier Individuen imitiert. Bei der *Mutation* wird der Rang einer zufällig gewählten Operation eines zufällig gewählten Individuums (einer Lösung der aktuellen Population) verändert. Dieser veränderte Plan wird anschließend so manipuliert, dass er wieder ein Plan, d.h. eine Lösung für das zugrundeliegende Schedulingproblem ist. Bei der *Kreuzung* wird eine zufällig bestimmte Teilmenge der Ränge zweier zufällig ausgewählter Individuen vertauscht. Anschließend werden auch in diesem Fall beide veränderte Pläne wieder zu Plänen gemacht. Auf diese Weise entstehen neue Individuen. Diese neuen Individuen werden dann in die nächste Generation übernommen, wenn sie gemäß einer Fitness-Funktion nicht weniger 'fit' sind, als ihre jeweiligen Vorgänger. Die Startpopulation wird dabei aus zufällig generierten Plänen gebildet. Mit Hilfe der **Autoalg**-Funktion (→ 7.3) ist es jedoch auch möglich sicherzustellen, dass die Startpopulation bereits

sehr gute Lösungen enthält, die mit selbstgewählten konstruktiven Heuristiken erzeugt wurden.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/GenAlg*** im Menü ***Algorithmen***.

Einstellungen

Populationsgröße: Hier wird angegeben, wie viele Lösungen zu einer Population gehören sollen.

Anzahl Generationen: Dieser Parameter bestimmt die Anzahl der Iterationen des Verfahrens.

Zufalls-Seed: Dieser Parameter dient als Eingabe für die zufällige Erzeugung der Startpopulation. Außerdem werden die zufälligen Entscheidungen in jedem Schritt beeinflusst. Wird das Verfahren ohne lokale Suchschritte durchgeführt, so kann es mit demselben Parameter exakt wiederholt werden.

Anzahl lokaler Suchschritte: Ein durch Mutation oder Kreuzung neu erzeugtes Individuum kann durch eine einfache **Lokale Suche** (→ 4.3.1) in der Regel leicht verbessert werden. Hier wird festgelegt, wie viele Iterationen der Lokalen Suche bei jeder neuen Lösung durchgeführt werden sollen.

Mutationswahrscheinlichkeit: Dieser Parameter gibt die Wahrscheinlichkeit an, mit der im aktuellen Iterationsschritt ein Individuum der aktuellen Population durch Mutation verändert wird.

Rekombinationswahrscheinlichkeit: Dieser Parameter gibt die Wahrscheinlichkeit an, mit der im aktuellen Iterationsschritt zwei Individuen der aktuellen Population durch Kreuzung (cross over) verändert werden.

Erzeugung der Anfangspopulation: Hier wird festgelegt, welche Art von Plänen für die Startpopulation zufällig erzeugt werden sollen.

- **RANDOM_ORDER:** Es werden Pläne generiert, deren Schedules aktiv oder non-delay sein können.
- **ACTIVE_DISPATCH:** Es werden ausschließlich Pläne mit aktiven Schedules generiert.
- **NON_DELAY_DISPATCH:** Es werden ausschließlich Pläne mit non-delay Schedules generiert.

Lokale Suche: Hier wird die Nachbarschaft festgelegt, in der neue Lösungen lokal verbessert werden sollen. Bei der Einstellung *(disabled)* wird keine lokale Suche durchgeführt.

- **API**
- **SHIFT**
- **PI**

- **CR_API**
- **BL_API**
- **CR_SHIFT**
- **BL_SHIFT**
- **3_CR**
- **(disabled).**

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden genetische Algorithmen wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE ga
  long POP_SIZE 20
  long NUM_GEN 100
  string INIT RANDOM_ORDER
  string FITNESS OBJECTIVE
  double M_PROB 0.25
  double C_PROB 0.35
  long SEED 1234567890
  long IMPR_STEPS 10
  string L_IMPR (disabled)
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für alle genetischen Verfahren ga aufgerufen werden. Unbedingt notwendig sind danach Angaben zu der **Populationsgröße**, der **Anzahl der Generationen**, der Methode zur **Erzeugung der Anfangspopulation** (INIT), der **Mutationswahrscheinlichkeit** (M_PROB) und der **Rekombinationswahrscheinlichkeit** (C_PROB). Optional ist die Vorgabe von Parametern für den **Zufalls-Seed**, die **Anzahl lokaler Suchschritte** und die Auswahl der Nachbarschaft der **Lokalen Suche**.
Zusätzlich kann durch den Aufruf eines genetischen Verfahrens mit Hilfe der Autoalg-Eingabedatei auch Einfluss auf die Wahl der **Fitness-Funktion** (FITNESS) genommen werden. Diese bestimmt, welche Individuen in die nächste Generation übernommen werden. Zur Auswahl stehen dabei zwei Optionen: Mit OBJECTIVE wählt man den jeweiligen Zielfunktionswert als Fitness-Funktion. Daneben kann auch SUM_Ci2 gewählt werden.

4.3.6 Beam Ant Colony Verfahren

Ant colony Optimierung ist eine Metaheuristik für schwer zu lösende Optimierungsprobleme. Basierend auf einem wahrscheinlichkeitstheoretischen Konstruktionsmechanismus wird durch Baumsuche eine Lösung konstruiert. Dabei entsteht bei Verwendung von Beamsearchverfahren auf diesem Baum ein hybrider Algorithmus, der vorliegende Algorithmus folgt den Ideen von Blum [4].

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Beam Ant Colony*** im Menü ***Algorithmen***.

Einstellungen

Anzahl der Beam-Extensions: Hier werden die maximal pro Erweiterungsschritt einzufügenden Operationen angegeben. Dieser Wert wird nur berücksichtigt, wenn die Bestimmung der Beam Extension auf FIXED gesetzt wurde.

Anzahl der Durchläufe: Anzahl der Einzelschritte. Da die Ameisen nacheinander gestartet werden, entspricht diese Zahl auch der Anzahl der Ameisen in der Gesamtpopulation.

Konvergenzfaktor für Neustart: Dieser Wert gibt eine untere Schranke für den Konvergenzfaktor an, ab dem eine Neuberechnung der Pheromonwerte angeregt wird.

evaporationRate: Dieser Parameter bezeichnet den Einfluss, den ein neuer bester Plan zur Berechnung der Pheromonwerte hat. Dieser Wert sollte zwischen 0 und 1 liegen.

Einfluss der Earliest Completion Time: Gilt nur für **Strategie der Auswahl der Extensions** = SORTED.

Einfluss der Pheromone Werte: Gilt nur für **Strategie der Auswahl der Extensions** = SORTED.

Einfluss des Zufalls: Gilt nur für **Strategie der Auswahl der Extensions** = SORTED Zielfunktionswert.

Bestimmung der Beam Extensions:

- **MED:** Anzahl der freien Operationen / 2.
- **LDS:** Zu Beginn wie MED, ab der zweiten Hälfte auf 2 festgelegt.
- **FIXED:** Feste Anzahl.

Strategie der Auswahl der Extension:

- **ORIGINAL:** Entspricht der in der Literatur beschriebenen Methode.
- **SORTED:** Hier werden die einzufügenden Operationen gewichtet und entsprechend dieses Wertes sortiert.

Vorauswahl der Erweiterungen:

- **NONE:**
- **ACTIVE:** Es werden nur aktive Pläne berücksichtigt.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) werden Beam Ant Colony-Verfahren wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE beam_aco
  long BEAM_WIDTH 32
  double UPPER_BOUND 10000000
  double LOWER_BOUND 0
  string EXTENSION_STRATEGY FIXED
  long STEPS 2500
  double CONVERGENCE_FACTOR 0.98
  long FIXED_KEXT 1
  string APPEND_STRATEGY SORTED
  string PRE_SELECTION ACTIVE
  double WEIGHT_EST 1.8
  double WEIGHT_TIJ 2
  double WEIGHT_RAND 1
  double EVAPORATION_RATE 0.3
</CONTROLPARAMETERS>
```

Die Reihenfolge, in der die Parameter aufgerufen werden, ist dabei irrelevant. Bei optionalen Parametern ist hier jeweils der default-Wert angegeben.
Als ausführbare Datei muss für Beam Ant Colony-Verfahren beam_aco aufgerufen werden. Optional können anschließend die folgenden Parameter aufgerufen werden: **Beamweite**, **obere Schranke**, **untere Schranke**, **Bestimmung der Beam-Extensions** (EXTENSION_STRATEGY), **Anzahl der Durchläufe** (STEPS), **Konvergenzfaktor** für Neustart, **Anzahl der Beam-Extensions** (FIXED_KEXT), Strategie zur **Auswahl der Extensions** (APPEND_STRATEGY), **Vorauswahl der Erweiterungen**, **Einfluss der Earliest Completion Time** (WEIGHT_EST), **Einfluss der Pheromone Werte** (WEIGHT_TIJ), und **Einfluss des Zufalls**.

4.4 Exakte Algorithmen zum Lösen spezieller Probleme

4.4.1 Brucker's Branch & Bound Algorithmus

Dieser exakte Algorithmus wurde an der Universität Osnabrück in der Arbeitsgruppe von Peter Brucker entwickelt und implementiert. Es gibt jeweils eine Variante für **open-shop Probleme** und eine für **job-shop Probleme**.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Exakte Verfahren/Brucker's Open-Shop B&B*** bzw. ***Exakte Verfahren/Brucker's Job-Shop B&B*** im Menü ***Algorithmen***.

Einstellungen

Für den **open-shop** Algorithmus muss eine Auswahl einer Heuristik zur Generierung von Plänen getroffen werden. Für den job-shop Algorithmus stehen keine Optionen zur Verfügung.

Heuristik: Der Algorithmus kann verschiedene Heuristiken zum Erzeugen oder vervollständigen von Plänen anwenden. (Es existieren mehr als die beiden hier aufgeführten Heuristiken. Diese sind jedoch deaktiviert, da sie nicht mit unvollständigen Operationenmengen funktionieren.)

- **LB_PREC_RULE:** Diese Heuristik basiert auf einer Vorrangregel, die sich aufgrund der Berechnung einer unteren Schranke ergibt.
- **LB_PREC_RULE_VAR:** Diese Heuristik ist eine Variation der Obigen.

Aufruf in der Autoalg-Eingabedatei

In einer Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird der Branch & Bound Algorithmus von Brucker folgendermaßen aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE os_bb_wo
  string HEURISTIC MIN_MATCHING
</CONTROLPARAMETERS>
```

Als ausführbare Datei muss für **open-shop** Probleme os_bb_wo, und für **job-shop** Probleme js_bb_br aufgerufen werden. Für open-shop Probleme kann danach die Wahl der **Heuristik** bestimmt werden. Für job-shop Probleme existieren keine weiteren Parameter.

4.4.2 Zweimaschinenprobleme

Für Probleme mit 2 Maschinen liefert die Johnsonsche Regel für das **flow-shop** Problem mit Makespanminimierung und deren Erweiterung auf den **job-shop** und **open-shop** Fall eine optimale Lösung, also gilt im Problemtyp $\alpha \in \{F2, J2, O2\}$.

Aufruf

Nach Eingabe von **Problemtyp** (wähle unter α neben F, J oder O auch ***gegebene Maschinenzahl***, denn diese hat als Startwert 2) und erforderlichen **Parametern** wählt man ***Exakte Verfahren/Johnsonsche Regel*** im Menü ***Algorithmen***.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die Johnsonsche Regel wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE johnson
</CONTROLPARAMETERS>
```

Für ein $O2 \mid\mid C_{max}$ Problem liefert auch die LAPT-Regel (longest alternating processing times) eine optimale Lösung.

Aufruf

Nach Eingabe von **Problemtyp** (beachte: $\alpha = O2$) und erforderlichen **Parametern** wählt man ***Exakte Verfahren/LAPT-Regel*** im Menü ***Algorithmen***.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die LAPT-Regel wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE pr
</CONTROLPARAMETERS>
```

4.4.3 Einmaschinenproblem mit Minimierung der Anzahl der verspäteten Jobs

Dieser Algorithmus minimiert in Einmaschinenumgebungen die Anzahl der verspäteten Aufträge. Er findet eine optimale Lösung in Polynomialzeit.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Exakte Verfahren/Eine Maschine - Summe Ui*** im Menü ***Algorithmen***.

Aufruf in der Autoalg-Eingabedatei

In einer Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird der Einmaschinenalgorithmus folgendermaßen aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE single_sum_ui
</CONTROLPARAMETERS>
```

Als ausführbare Datei muss single_sum_ui aufgerufen werden. Es werden keine Parameter erwartet.

4.4.4 Open-shop Problem mit Makespanminimierung und pmtn (Unterbrechung ist erlaubt)

Implementiert ist der Algorithmus von Gonzales/Sahni [14]. Da der LiSA-Kern aber noch keine Ganttdiagramme mit Unterbrechung darstellen kann und auch ein Plan, ein Plangraph und ein Schedule mit pmtn noch nicht in der grafischen Oberfläche enthalten sind, wird das Ergebnis in eine Datei geschrieben. Man nutzt die LiSA Oberfläche, um den **Problemtyp** und die erforderlichen **Parameter** einzugeben. Dann ruft man ***Exakte Verfahren/Gonzales & Sahni (Reasearch)*** auf. LiSA speichert die Ergebnisse in der Datei **algo_out.xml** in das Verzeichnis `~/.lisa/proc`. An jeder Stelle (ij) der Matrizen stehen Informationen zu den Teiloperationen, in die (ij) zerfällt. Nähere Informationen zur Erweiterung des Block-Matrizen Modells auf Probleme mit Unterbrechungen siehe Bräsel/Hennes [6].

4.5 Heuristische Algorithmen für spezielle Probleme

4.5.1 Die Shifting-Bottleneck Heuristik

Die Shifting-Bottleneck Heuristik erzeugt eine Näherungslösung für das **job-shop** bzw. **flow-shop** Problem mit Makespanminimierung durch das sukzessive Lösen von Einmaschinenproblemen. Im ersten Schritt wird das Gewicht eines kritischen Weges im zugehörenden disjunktiven Graphen, in dem alle ungerichteten Kanten entfernt wurden, berechnet. Dieser Wert ist eine untere Schranke LB für die Gesamtbearbeitungszeit C_{max}. Schritt für Schritt wird jetzt für jede Maschine ein Einmaschinenproblem mit Bereitstellungszeiten der Aufträge (Heads der Operationen auf der betrachteten Maschine), Fälligkeitsterminen (LB minus Tails der Operationen auf der betrachteten Maschine), Vorrangbedingungen der Aufträge (Vorrangbedingungen der Operationen auf der betrachteten Maschine aus dem jeweils aktuellen disjunktiven Graphen ohne disjunktive Kanten) und Minimierung von L_{max} gelöst. Die exakte Lösung dieses Problems wird als organisatorische Reihenfolge der Operationen auf der betrachteten Maschine angesetzt. Daneben wird nach der Festlegung der organisatorischen Reihenfolge auf einer Maschine versucht, bereits festgelegte organisatorische Reihenfolgen weiter zu verbessern, man shiftet den Bottleneck.

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/ShiftingBottleneck*** im Menü ***Algorithmen***.

Einstellungen

Speed: Mit dieser Auswahl sind eine schnelle Variante, die aber auf unzulässige Pläne führen kann, oder eine langsamere Variante, die die vollständige Heuristik enthält, verfügbar.

- **FAST** Die schnelle Variante berücksichtigt nicht den Vorranggraph der Operationen bei der Lösung der Einmaschinenprobleme. Dadurch können Kombinationen aus Technologie und Organisation erzeugt werden, die unzulässig sind, also Zyklen enthalten, was LiSA als Fehlermeldung zurückgibt.
- **SLOW** Hier ist die Shifting-Bottleneck Heuristik vollständig implementiert.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die Shifting-Bottleneck Heuristik wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE bottle
  string SPEED SLOW
</CONTROLPARAMETERS>
```

Testrechnungen mit zufällig erzeugten technologischen Reihenfolgen haben erst bei Problemen mit mehr als 20 Maschinen und 20 Aufträgen in der FAST- Variante zu Zyklen geführt.

4.5.2 Die flow-shop Heuristik

In dieser Heuristik für das **flow-shop** Problem wird aus der Menge der Permutationspläne (auf allen Maschinen liegt die gleiche organisatorische Reihenfolge vor) eine Näherungslösung bestimmt. Dafür wird eine Insertionreihenfolge der Jobs nicht steigend nach ihren Gesamtbearbeitungszeiten bestimmt. Man beginnt mit dem ersten Job dieser Reihenfolge und ordnet den zweiten direkt vor dem ersten und direkt nach ihm an. In der besten Variante hinsichtlich des Makespan wird der dritte Auftrag auf Position 1, 2 und 3 eingefügt, usw. (Beam-Insert mit der Weite 1 auf der Menge aller Permutationspläne).

Aufruf

Nach Eingabe von **Problemtyp** und erforderlichen **Parametern** wählt man ***Heuristische Verfahren/Flow Shop Heuristic*** im Menü ***Algorithmen***.

Aufruf in der Autoalg-Eingabedatei

In der Eingabedatei für die **Autoalg-Funktion** (→ 7.3) wird die flow-shop Heuristik wie folgt aufgerufen:

```
<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE fsheur
</CONTROLPARAMETERS>
```

Als ausführbare Datei muss fsheur angegeben werden. Es werden keine weiteren Parameter erwartet.

Kapitel 5

Ausgabe

Sobald LiSA einen Schedule berechnet hat oder eine Datei mit einem Schedule geöffnet hat, zeigt LiSA im Hauptfenster standardmäßig das zugehörende Gantt-Diagramm. Dann lassen sich im Menüpunkt ***Ansicht*** die folgenden Informationen zum aktuellen Schedule aufrufen:

- Parameter
- Plan → 5.1
- Plangraph → 5.2
- Schedule → 5.3
- Gantt-Diagramm → 5.4

Durch Aufruf des Menüpunktes von ***Datei/Drucken*** wird der aktuelle Inhalt des Hauptfensters ausgedruckt. Daneben gibt es die Möglichkeit, die aktuellen Ergebnisse als xml-Datei abzuspeichern, siehe **Ergebnisdatei** → 5.5.

5.1 Plan (Sequence)

Ein Plan ist eine zulässige Kombination aus den technologischen Reihenfolgen (machine order) der Aufträge und den organisatorischen Reihenfolgen (job order) auf den Maschinen. Er wird durch die Rangmatrix der Knoten (Operationen) des (azyklischen) Sequencegraphen (siehe 5.2) beschrieben. Der Rang eines Knotens v in einem azyklischen Digraphen ist definiert als Knotenanzahl eines längsten Weges zum Knoten v, wobei v mitgezählt wird.

Es sei bemerkt, dass jeder Plan ein lateinisches Rechteck ist, das die sogenannte Sequencebedingung erfüllt: Steht an der Position (i, j) im lateinischen Rechteck eine Zahl $a > 1$, dann existiert entweder in Zeile i oder in Spalte j oder in beiden der Eintrag $a - 1$. Ein lateinisches Rechteck $LR[m, n, r]$ ist eine Matrix vom Format $m \times n$ mit Einträgen aus der Menge $B = \{1, ..., r\}$, wobei jede Zahl aus B in jeder Zeile und Spalte höchstens einmal vorkommt.

Nähere Informationen: siehe Block-Matrizen-Modell → 2.3.2

Aufruf

Nach Eingabe einer Probleminstanz und erster Anwendung eines Algorithmus wird standardmäßig der entstehende Schedule durch ein Gantt-Diagramm dargestellt. Dann wird nach Wahl des Menüpunktes ***Ansicht/Plan*** der Plan im Hauptfenster angezeigt.

5.2 Plangraph (Sequencegraph)

Der Plangraph (Sequencegraph) $G(MO, JO)$ ist ein azyklischer Digraph mit den Operationen als Knoten. Die gerichteten Kanten (Bögen) entsprechen den direkten Vorrangbedingungen der Operationen in den technologischen und organisatorischen Reihenfolgen der Operationen, d.h. von der Operation (ij) gibt es einen Bogen zur Operation (kl), wenn entweder $i = k$ und der Auftrag J_i geht nach der Bearbeitung auf M_j direkt zur Bearbeitung auf M_l über oder $j = l$ und auf der Maschine M_j wird nach dem Auftrag J_i direkt der Auftrag J_k bearbeitet gilt.
Nähere Informationen: siehe Block-Matrizen Modell → 2.3.2

Aufruf

Nach Eingabe einer Probleminstanz und erster Anwendung eines Algorithmus wird standardmäßig der entstehende Schedule durch ein Gantt-Diagramm dargestellt. Dann wird nach Wahl des Menüpunktes ***Ansicht/Plangraph*** der Plangraph im Hauptfenster angezeigt.

Extras

Bei Minimierung der Gesamtbearbeitungszeit sind die Bögen des kritischen Weges (bzw. der kritischen Wege) rot eingefärbt.

5.3 Schedule (Zeitplan)

Werden in einem Plangraphen die Operationen mit der zugehörenden Bearbeitungszeit gewichtet, so läßt sich ein Schedule (Zeitplan der Abarbeitung aller Operationen) konstruieren. Die Menge der zu einem Plangraphen gehörenden Schedules hat unendlich viele Elemente, aber zu einem Schedule gehört ein eindeutiger Plan. Ein Schedule heißt:

- **semiaktiv**, wenn alle Operationen so früh wie möglich bearbeitet werden, wobei der zugrundeliegende Plan eingehalten wird.
- **aktiv**, wenn es keine Stillstandszeit auf den Maschinen gibt, in der eine später angeordnete Operation vollständig bearbeitet werden kann.
- **non-delay**, wenn es keinerlei Verzögerung der Abarbeitung der Operationen gibt, d.h. wenn zu einem Zeitpunkt eine Maschine frei ist und ein Auftrag zur Abarbeitung auf dieser Maschine wartet, muss mit der Bearbeitung begonnen werden.

Jeder non-delay Schedule ist aktiv und jeder aktive Schedule ist semiaktiv, die Umkehrung gilt nicht.

In LiSA werden semiaktive Schedules durch die Matrix $C = [c_{ij}]$ beschrieben, wobei c_{ij} die Fertigstellungszeit der Operation (ij) bezeichnet.
Nähere Informationen: siehe Block-Matrizen Modell → 2.3.2

Aufruf

Nach Eingabe einer Probleminstanz und erster Anwendung eines Algorithmus wird standardmäßig der entstehende Schedule durch ein Gantt-Diagramm dargestellt. Dann wird nach Wahl des Menüpunktes ***Ansicht/Schedule*** der Schedule im Hauptfenster angezeigt.

Extras

Bei Minimierung der Gesamtbearbeitungszeit sind die c_{ij} rot eingefärbt, die zu den Operationen (ij) gehören, durch die mindestens ein kritischer Weg geht.

5.4 Gantt-Diagramm

Das Gantt-Diagramm (benannt nach Henry Laurence Gantt, 1861-1919) veranschaulicht den zeitlichen Ablauf der Bearbeitungen der Operationen, d.h. es visualisiert einen Schedule.

Aufruf

Nach Eingabe einer Probleminstanz und erster Anwendung eines Algorithmusses wird standardmäßig der entstehende Schedule durch ein Gantt-Diagramm dargestellt. Es wird darüberhinaus nach Wahl des Menüpunktes ***Ansicht/Gantt Diagramm*** im Hauptfenster angezeigt. Verschiedene Darstellungen des Gantt-Diagramms können im Menüpunkt ***Optionen/Gantt Diagramm*** eingestellt werden, sobald ein Schedule vorhanden ist.

Einstellungen

Orientierung:

- **Maschinenorientiert:** Die Operationen der Aufträge erscheinen als Balken über der Zeitachse (x-Achse) auf den Maschinen (y-Achse).
- **Auftragsorientiert:** Die Operationen auf den Maschinen erscheinen als Balken über der Zeitachse (x-Achse) für die Aufträge (y-Achse). Hier werden auch eventuell vorhandene Bereitstellungs- und Fälligkeitstermine dargestellt.

Besondere Einstellungen:

- **Keine:** Standard: Die Operationen werden mit bis zu 23 gut unterscheidbaren Farben eingefärbt.
- **Kritischer Weg:** Bei Makespanminimierung werden die Operationen, die zu einem kritischen Weg gehören, rot eingefärbt, alle anderen Operationen erscheinen grau.
- **Definiere Farben für einige Aufträge bzw. Maschinen:** Es können Farben für maximal 5 Aufträge bzw. Maschinen explizit ausgewählt werden.

Extras

Manipulationen des Gantt-Diagramms:

- **Zoom:** Die Ansicht kann gezoomt werden. Dazu klicken Sie auf das Lupen-Symbol oder aktivieren den Zoom-Modus im Menü ***Extras***. Ziehen Sie dann mit der Maus ein Rechteck auf, um dieses zur neuen Ansicht zu machen. Sie können sich im gewählten Zoom-Modus durch das Gantt Diagramm scrollen. Um wieder das ganze Diagramm zu sehen, deaktivieren Sie den Zoom-Modus.

- **Plan-Editor:** Ein Doppelklick auf eine Operation startet die **Manipulation von Plänen und Schedules** (→ 6.2), mit der es möglich ist, die gewählte Operation in der zugehörenden technologischen bzw. organisatorischen Reihenfolge zu verschieben.

Nähere Informationen, siehe Manipulation von Plänen und Schedules → 6.2

5.5 Ergebnisdatei

Durch Aufruf des Menüpunktes von ***Datei/Speichern als*** werden die folgenden Informationen in einer xml-Datei abgespeichert, wobei man den Namen der Datei und den Abspeicherungsort frei wählen kann:

- Problemtyp in der $\alpha \mid \beta \mid \gamma$ - Notation;
- n (Anzahl der Aufträge) und m (Anzahl der Maschinen);
- Matrix der Bearbeitungszeiten;
- Operationenmenge (mit Hilfe einer Boolschen Matrix);
- Plan als lateinisches Rechteck mit Sequencebedingung;
- Schedule als Matrix der Fertigstellungszeiten aller Operationen.

Diese Datei kann auch wieder als Eingabedatei für LiSA verwendet werden.
Nähere Informationen zum Dateiformat: Das File Format in LiSA → 2.5

Kapitel 6

Extras

LiSA enthält einige interessante interne Bausteine, auf die in diesem Kapitel eingegangen wird.

6.1 Komplexitätsstatus eines Problems

LiSA ist in der Lage, den Komplexitätsstatus eines Problems in der $\alpha \mid \beta \mid \gamma$ - Notation anzugeben. Die Bestimmung der Komplexität wird durch Analyse einer BibTeX-Datenbank (Datei classify.bib) durchgeführt, die auf der folgenden Sammlung von Resultaten für Schedulingprobleme basiert: Complexity results of scheduling problems, siehe: http://www.mathematik.uni-osnabrueck.de/research/OR/class/.

Aufruf

Sobald ein Problemtyp eingegeben ist, kann man den Menüpunkt ***Extras/Problemklassifikation*** anwählen. Im Fenster Problemklassifikation wird angegeben, ob das Problem polynomial lösbar, pseudo-polynomial lösbar, NP-schwer oder NP-schwer im strengen Sinne ist.
Wenn die Komplexität des betrachteten Problems bekannt ist, wird ein Hinweis auf die entsprechende Literaturquelle gegeben, die die Aussage belegt. Für die Ausgabe der entsprechenden vollständigen Literaturquellen muss der Button ***Vollstaendige Literaturquellen*** gedrückt werden.

6.2 Manipulation von Plänen und Schedules

LiSA ermöglicht es Ihnen, einen bereits berechneten Plan oder Schedule nachträglich zu manipulieren, indem Sie einzelne Operationen in der jeweiligen technologischen und/oder organisatorischen Reihenfolge verschieben.

Aufruf

Wählen Sie eine zu manipulierende Operation aus, indem Sie doppelt auf den entsprechenden Balken im Gantt-Diagramm klicken. Die gewählte Operation $(i\,j)$ wird mit einem schwarzen Rand markiert und es erscheint ein Fenster mit den Manipulationsmöglichkeiten.

Einstellungen

Manipulationsmöglichkeiten für die ausgewählte Operation *(i j)*:

Zeichen	Bedeutung
Quelle	Die Operation $(i\,j)$ wird zu einer Quelle im Plangraphen, d. h. sie hat keine Vorgängeroperationen.
Senke	Die Operation $(i\,j)$ wird zu einer Senke im Plangraphen, d. h. sie hat keine Nachfolgeroperationen.
MO ⏮	Die Operation $(i\,j)$ wird erste Operation in der technologischen Reihenfolge des Jobs i.
MO ◀	Die Operation $(i\,j)$ wird in der technologischen Reihenfolge von Job i um eine Position nach vorn verschoben.
MO ⏭	Die Operation $(i\,j)$ wird letzte Operation in der technologischen Reihenfolge von Job i.
MO ▶	Die Operation $(i\,j)$ wird in der technologischen Reihenfolge von Job i um eine Position nach hinten verschoben.
JO ⏫	Die Operation $(i\,j)$ wird erste Operation in der organisatorischen Reihenfolge auf Maschine j.
JO ▲	Die Operation $(i\,j)$ wird in der organisatorischen Reihenfolge auf Maschine j um eine Position nach vorn verschoben.
JO ⏬	Die Operation $(i\,j)$ wird letzte Operation in der organisatorischen Reihenfolge auf Maschine j.
JO ▼	Die Operation $(i\,j)$ wird in der organisatorischen Reihenfolge auf Maschine j um eine Position nach hinten verschoben.
➲	Es wird der Plan wiederhergestellt, der vor dem Öffnen des Manipulationsfensters vorlag.

ACHTUNG: Wenn die Verschiebung der Operation einen Zyklus im Graphen $G(MO, JO)$ erzeugt, öffnet sich ein Fenster mit der Fehlermeldung: *cycle: modification not possible.*

Im unteren Teil des Manipulationsfensters sind die unmittelbaren Vorgänger und Nachfolger der ausgewählten Operation $(i\,j)$ zu sehen. Gleichzeitig werden die Bearbeitungszeit p_{ij} und die Fertigstellungszeit c_{ij} angezeigt.

6.3 Irreduzibilitätstest

Dieser Test löst verschiedene Aufgabenstellungen aus der Irreduzibilitätstheorie, ausführliche Informationen sind in der Dissertation von Willenius [26] enthalten. Er ist für open, job und flow shop Probleme mit Bereitstellungszeiten der Jobs und mit den regulären Zielfunktionen Cmax, Lmax, SumCi, SumWiCi, SumUi, SumWiUi, SumTi und SumWiTi anwendbar.
Ein Plan S^* reduziert einen Plan S, wenn S^* für jede Wahl der Bearbeitungszeiten keinen schlechteren Zielfunktionswert als S liefert. Zwei Pläne sind zueinander ähnlich, wenn sie für jede Wahl von Bearbeitungszeiten den gleichen Zielfunktionswert besitzen. Ein Plan S^* reduziert einen Plan S streng, wenn sie nicht ähnlich zueinander sind und wenn S^* den Plan S reduziert. Ein Plan S heißt irreduzibel, wenn es keinen Plan S^* gibt, der S echt reduziert. Damit existiert in der Menge aller irreduziblen Pläne ein global optimaler Plan für jede Wahl der Bearbeitungszeiten.

Aufruf

Wenn LiSA den Problemtyp und alle Parameter kennt und bereits ein Plan erzeugt wurde, läßt sich der Algorithmus im Menüpunkt ***Heuristische Verfahren/Irreduzibilitätstest*** des Menüs ***Algorithmen*** aufrufen.

Einstellungen

Erzeuge Pläne: Es wird eingegeben, welche Pläne erzeugt werden sollen:

- **SIMILAR** Es werden alle zum Ausgangsplan ähnlichen Pläne erzeugt.
- **ALL_REDUCING** Es werden alle Pläne erzeugt, die den Eingabeplan streng reduzieren und zwar genau einer aus jeder Ähnlichkeitsklasse. Wenn kein solcher Plan existiert, wird eine LiSA-Fehlermeldung ausgegeben: Warning: Plan is irreducible.
- **ITERATIVE_REDUCING** Sobald ein Plan gefunden wurde, der den ursprünglichen Plan streng reduziert, wird der Algorithmus unterbrochen und mit dem neuen Plan erneut gestartet. Dies wird fortgeführt, bis ein irreduzibler Plan gefunden wurde.

  Sollte der Ausgangsplan irreduzibel sein, wird eine LiSA-Fehlermeldung ausgegeben: Warning: Sequence is irreducible.

Gib folgende Pläne zurück: Eine Abspeicherung erfolgt in der Datei *algo_out.xml* im Verzeichnis `~/.lisa/proc`. In LiSA wird das Ergebnis unter ***Bearbeiten/Liste von Plaenen*** angezeigt.

- **ALL** Alle erzeugten Pläne werden zurückgegeben, wobei die Anzahl im Zusammenhang mit der ALL_REDUCING Option sehr groß sein kann.
- **ONLY_IRREDUCIBLE** Nur irreduzible Pläne werden zurückgegeben.

Erzeuge Pläne in zufälliger Reihenfolge: Auswahlmöglichkeiten:

- **YES** Pläne werden vom Algorithmus in zufälliger Reihenfolge erzeugt. Dies ist nur interessant im Zusammenhang mit der ITERATIVE_REDUCING Option, um bei verschiedenen Aufrufen des Algorithmus verschiedene Resultate zu erhalten.
- **NO** Pläne werden vom Algorithmus in einer effizienten Reihenfolge erzeugt.

Kapitel 7

Zur externen Arbeit mit LiSA

7.1 Algorithmenmodule

Alle Algorithmen zur Lösung von Schedulingproblemen in LiSA sind in externe Module ausgelagert. Es handelt sich dabei um eigenständige ausführbare Programme, die nach der Auswahl im Menü ***Algorithmen*** von LiSA aufgerufen werden. Sie können aber auch manuell über die Kommandozeile ausgeführt werden.

Aufbau eines Moduls

Ein Modul in LiSA besteht im Wesentlichen aus fünf einzelnen Dateien:

- einer XML-Datei vom **Dokumenttyp** (→ 2.5) `algorithm` zur Algorithmenbeschreibung,
- einer Hilfedatei im HTML-Format,
- und schließlich dem Algorithmus selbst in Form eines eigenständigen Programms,

wobei die ersten beiden Dateien jeweils in einer deutschen und englischen Version vorliegen müssen.
Das XML-Dokument dient dabei zur Einbindung des Moduls in das Hauptprogramm. Es befindet sich in den Unterordnern `data/alg_desc/language/german` und `data/alg_desc/language/english`. Dort wird unter anderem festgelegt, welche Problemtypen ein Algorithmus lösen kann, ob er auf einer bereits gefundenen Lösung aufbaut oder ob er ein Problem heuristisch oder exakt löst. Zudem können hier Parameter deklariert werden, die dem Algorithmus übergeben werden. Der genaue Aufbau eines solchen XML-Dokuments ist im Anhang A beschrieben.
Die Hilfedatei dient zur Beschreibung des Algorithmus in einfacher Form. Sie sollte kurz auf die Lösungsstrategie eingehen, die lösbaren Problemtypen aufzählen und die Programmparameter beschreiben. Sie befindet sich im Unterordner `doc/lisa/english/algorithm` und `doc/lisa/german/algorithm`.
Der Algorithmus selbst liegt im Unterordner `bin`.

Kommandozeilenschnittstelle

Das Programm, das den Algorithmus ausführt, nimmt genau zwei Kommandozeilenparameter entgegen: eine Eingabedatei und eine Ausgabedatei. Die Eingabedatei muss eine LiSA-Probleminstanz im XML-Format enthalten. Sie kann entweder vom **Dokumenttyp** (→ 2.5)

`instance` oder `solution` sein. Die Ausgabedatei wird vom Algorithmenmodul erzeugt und enthält die vom Algorithmus berechnete Lösung des Problems.

Übergabe von Parametern

Kapitel 4 beschreibt zu jedem Algorithmus einen Satz von Parametern, die dessen Verhalten beeinflussen. Die Übergabe der Parameter erfolgt mit Hilfe der Eingabedatei. Zu diesem Zweck steht das `<controls>`-Element zur Verfügung, das sich in die Dokumente des Typs `instance` und `solution` einfügen lässt. Es enthält einzelne Parameterbelegungen in Form von `<parameter>`-Elementen. Mit diesen erfolgt eine Wertzuweisung für einen Parameter durch Angabe von Datentyp (`integer`, `real` oder `string`), Parametername und dem Wert. Ein `<controls>`-Bock für einen Aufruf des Algorithmus **Iterative Suche** (→ 4.3.1) kann beispielsweise so aussehen:

```
<controls>
  <parameter type="string" name="METHOD" value="IterativeImprovement" />
  <parameter type="string" name="NGBH" value="k_API" />
  <parameter type="integer" name="k" value="1" />
  <parameter type="integer" name="STEPS" value="1" />
  <parameter type="integer" name="NUMB_STUCKS" value="214748000" />
  <parameter type="real" name="ABORT_BOUND" value="-214748000" />
</controls>
```

In `instance`-Dokumenten müssen `<controls>`-Böcke direkt nach dem `<values>`-Element eingefügt werden. In `solution`-Dokumenten stehen sie ebenfalls hinter dem `<values>`-Element, noch vor dem ersten `<schedule>`-Element.

Ausgabe von Laufzeitinformationen

Algorithmenmodule können mit Hilfe ihrer Standardausgabe Informationen an das Hauptprogramm ausgeben. Dazu stehen die vier Schlüsselworte `PID=`, `OBJECTIVE=`, `WARNING:` und `ERROR:` bereit.

`PID=x` teilt dem Hauptprogramm die Prozess-ID `x` des laufenden Moduls mit. Diese Information sollte möglichst zu Beginn der Ausführung ausgegeben werden.

`OBJECTIVE=x` gibt eine Meldung über den aktuellen Rechenstatus zurück. Als Ausgabewert kann z.B. der aktuelle Zielfunktionswert dienen. Diese Werte werden dann von der LiSA-Oberfläche während der Berechnung grafisch dargestellt. Die erste Ausgabe setzt dabei die Größe des Anzeigefensters, alle weiteren Meldungen sollten also wenn möglich unter dieser Grenze bleiben.

`WARNING:msg` zeigt ein Meldungsfenster mit der Warnungsmeldung `msg`. Das Algorithmenmodul setzt danach seine Ausführung fort.

`ERROR:msg` zeigt ein Meldungsfenster mit der Fehlermeldung `msg` und bricht die Ausführung des Algorithmus ab.

Manuelles Aufrufen eines Algorithmus

Soll ein Algorithmus unabhängig von der LiSA-Oberfläche ausgeführt werden, kann dies also duch den Aufruf der entsprechenden ausführbaren Datei im Unterordner `bin` geschehen. Dazu muss eine Eingabedatei erstellt werden, die das zu lösende Problem enthält. Solche Dateien lassen sich einfach erzeugen, indem ein in LiSA erstelltes Problem mit dem Menüpunkt ***Datei/Speichern als*** abgespeichert wird (→ 3.3). Es müssen dann jedoch noch die Parameter für den Algorithmus von Hand eingefügt werden (mindestens ein leeres `<controls />`-Tag, wenn alle Parameter mit den Standardwerten belegt werden sollen).
Für nähere Informationen zu den Parametern der einzelnen Algorithmen siehe Kapitel 4 und die XML-Referenz im Anhang A.

7.2 Einbinden externer Algorithmen

Zum Einfügen eines neuen Algorithmenmoduls genügt es, die im vorherigen Kapitel beschriebenen Dateien zu erstellen. Prinzipiell kann der Algorithmus in jeder beliebigen Programmiersprache implementiert werden, die in das Binärformat übersetzt werden kann. LiSA stellt jedoch speziell für C++ Klassen bereit, die die Arbeit mit Scheduling-Problemen erleichtern sollen und die Implementierung von Algorithmen vereinfachen. So lassen sich z.B. Problembeschreibung, Schedules und Parameter aus den XML-Dateien lesen und schreiben. Zudem werden grundlegende und auch spezielle Datentypen bereitgestellt, die für den Umgang mit Scheduling-Problemen hilfreich sind. Um diese Klassenbibliothek nutzen zu können, muss das LiSA-Quellcodepaket installiert sein.
Im folgenden Abschnitt wird davon ausgegangen, dass ein Algorithmus in C++ implementiert werden soll.
Neue Algorithmen sollten im Unterordner `src/algorithm` ein eigenes Verzeichnis bekommen, um den späteren Erstellungsvorgang zu vereinfachen. Dort müssen die folgenden Dateien erstellt werden:

- C++-Quellcode und -Headerdateien, die den Algorithmus implementieren.
- `Makefile`
- `Make.List`
- Die im Abschnitt 7.1 genannten XML- und HTML-Dateien. Diese werden beim Erstellungsvorgang in die jeweiligen Ordner verschoben.
- `Make.Depend` und `Make.Objects`, welche automatisch erstellt werden.

Quellcode

Der Algorithmus selbst ist als gewöhnliches Konsolenprogramm implementiert, das die im Kapitel 7.1 genannten Eingaben entgegennimmt und die daraus brechnete Lösung in eine Ausgabedatei schreibt. Als einführendes Beispiel sei hier auf den Algorithmus im Ordner `src/algorithm/sample` verwiesen, wo der grundlegende Aufbau einer Algorithmenimplementation bereits vorgefertigt ist. Es genügt meist, die Quellcodedatei `sample.cpp` zu übernehmen und die markierte Passage durch eigenen Code zu ersetzen. Zudem ist es natürlich möglich, eigenen Code in andere Dateien auszulagern.

Makefile

Der folgende Grundaufbau des Makefiles kann für C++-Programme direkt übernommen werden. Es muss lediglich der Programmname angepasst werden; dies ist der Name des Unterordners, in dem die Quellcodedateien des Algorithmus liegen.

```
# LiSA Sample Algorithm Makefile

# ----------------------------------------------------------------

# LiSA part: sample

PROGRAMNAME=sample

# ----------------------------------------------------------------

TOPPROGRAMPATH=../../..

# ----------------------------------------------------------------

include ../Make.Algorithm
```

Make.List

In dieser Datei werden alle C++-Quellcodedateien angegeben, die beim Erstellungsvorgang kompiliert und zur ausführbaren Moduldatei gelinkt werden. Neben den eigentlichen Quellcodedateien des Moduls müssen hier auch alle externen Abhängigkeiten angegeben werden, damit der Linker alle Symbole auflösen kann. Wird im Quellcode des Moduls z.B. die Klasse `LISA_Matrix` verwendet, muss die entsprechende Quellcodedatei `matrix.cpp` im Unterordner `src/basics` verlinkt werden. Die Pfadangabe erfolgt dabei immer relativ zum Modulordner, also muss letztendlich der Eintrag `../../basics/matrix.cpp` eingefügt werden. Die entsprechenden Quellcodedateien zu den Klassen lassen sich aus der Dokumentation zur LiSA-Klassenbibliothek entnehmen.

```
CXXSOURCES=\
  sample.cpp \
  ../../basics/list.cpp \
  ../../basics/matrix.cpp \
  ../../basics/pair.cpp \
  ../../lisa/ctrlpara.cpp\
  ../../lisa/lvalues.cpp \
  ../../lisa/ptype.cpp \
  ../../main/global.cpp \
  ../../misc/except.cpp \
  ../../misc/int2str.cpp \
  ../../scheduling/mo_jo.cpp \
  ../../scheduling/schedule.cpp
```

Make.Depend und Make.Object

Diese Dateien werden duch einen einmaligen Aufruf `make depend` im Modulverzeichnis aus der Datei `Make.List` erstellt. Wird die Datei `Make.List` verändert, müssen sie durch ein erneutes `make depend` aktualisiert werden.

XML- und HTML-Dokumente

Diese Dokumente müssen in einer englischen und einer deutschen Version vorliegen (die beiden Versionen des XML-Dokuments unterscheiden sich meist nur durch die Übersetzung des Algorithmennamens und der Parameter). Die englischen Versionen der Dokumente bekommen die Endungen `_english.xml` bzw. `_english.html` und die deutschen Versionen die Endungen `_german.xml` bzw. `_german.html`. Beim Erstellungsvorgang werden diese Endungen dann durch `.html` bzw. `.xml` ersetzt und die Dokumente an ihre Bestimmungsorte verschoben.

Erstellungsvorgang

Der Erstellungsvorgang kann mit Hilfe des Aufrufs `make` im Modulordner gestartet werden. Im Laufe dieses Vorgangs werden Quellcodedateien, die in `Make.list` angegeben sind, kompiliert und zu einer ausführbaren Datei gelinkt. Anschließend werden alle Moduldateien an ihre Bestimmungsorte kopiert (→ 7.1).

7.3 Automatisierter Algorithmenaufruf

Mit Hilfe des Programms `auto_alg`, das im Verzeichnis `LiSA/bin`. enthalten ist, kann der Algorithmenaufruf automatisiert werden. Um diese Funktion nutzen zu können, muss zunächst eine Eingabedatei erstellt werden, die anschließend als Parameter für den Programmaufruf `auto_alg` fungiert.
Dabei werden mit Hilfe des in LiSA enthaltenen Zufallsgenerators (siehe Taillard [24]) automatisch Instanzen von Schedulingproblemen erzeugt, für die bestimmte Algorithmen aufgerufen werden können. So kann z.B. ein und dasselbe Problem automatisch mit mehreren verschiedenen Algorithmen gelöst und die Ergebnisse anschließend verglichen werden.
Für jede Probleminstanz erzeugt der Aufruf von `auto_alg` eine XML-Datei mit der Probleminstanz und den Lösungen, die die aufgerufenen Algorithmen erzeugen. Daneben werden die Startwerte für die verwendete Zufallszahlerzeugung angegeben. Diese XML-Datei kann mit LiSA geöffnet werden (Button: Bearbeiten/Liste von Plaenen). Jede Lösung ist aufrufbar, aber eine weitere Verarbeitung einer speziellen Lösung löscht die Liste wieder.

ACHTUNG

Die Arbeit mit dem automatisierten Algorithmenaufruf muss sehr sorgfältig erfolgen, da die Eingabedatei nicht intern auf Richtigkeit überprüft wird. Insbesondere hat der Nutzer darauf zu achten, ob der aufgerufene Algorithmus überhaupt für den betrachteten Problemtyp zur Verfügung steht und ob die Parameter für die Probleminstanzerzeugung und die Algorithmen richtig sind (Gross- und Kleinbuchstaben und Leerzeichen beachten!). Eine weitere Fehlerquelle liegt in der Angabe der Anzahl der auszuführenden Algorithmen.

Erstellen einer Eingabedatei

Die Eingabedatei ist eine Textdatei (z.B. `~/test/myproblem.alg`), die den zu bearbeitenden Problemtyp, Parameter für den Zufallsgenerator und die abzuarbeitenden Algorithmen

definiert. Das folgende ausführliche Beispiel soll den Aufbau verdeutlichen. Es generiert zehn Probleminstanzen vom Typ O | r_i | $\sum w_i T_i$. Auf jede Instanz wird zunächst die Zufallsreihungsregel angewendet und anschließend die Lösung per iterativer Suche verbessert.

```
<PROBLEMTYPE>
  Lisa_ProblemType= { O / r_i / SumWiTi }
</PROBLEMTYPE>

<CONTROLPARAMETERS>
  long MINPT 1
  long MAXPT 99
  long MINRI 0
  long MAXRI 100
  long MINDD 200
  long MAXDD 1000
  double MINWI 0.0
  double MAXWI 2.0
  long M 10
  long N 20
  long TIMESEED 658496
  long MACHSEED 2465865
  long DDSEED 1123545
  long WISEED 8885564
  long RISEED 566945
  long NUMBERPROBLEMS 10
  long NUMBERALGORITHMS 3
  long RIDIMODE 0
  double TFACTOR 0.7
</CONTROLPARAMETERS>

<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE lower_bounds
</CONTROLPARAMETERS>

<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE dispatch
  string SCHEDULE ACTIVE
  string RULE RANDOM
</CONTROLPARAMETERS>

<CONTROLPARAMETERS>
  string AUTOALG_EXECUTABLE nb_iter
  string METHOD IterativeImprovement
  string NGBH k_API
  long k 1
  long STEPS 1000
  long NUMB_STUCKS 1000
```

```
  double ABORT_BOUND -1
  string TYPE RAND
</CONTROLPARAMETERS>
```

Der erste Block, der mit `<PROBLEMTYPE> ... </PROBLEMTYPE>` umschlossen ist, gibt den zu behandelnden Problemtyp in $\alpha \mid \beta \mid \gamma$ Notation an. Es können folgende Angaben gemacht werden (siehe auch **Problemtyp** $\rightarrow$ 2.2):

Parameter	**Mögliche Werte**
α	`O`, `J` oder `F`
β	`r_i` oder nichts (freilassen)
γ	`Cmax`, `Lmax`, `SumCi`, `SumWiCi`, `SumUi`, `SumWiUi`, `SumTi` oder `SumWiTi`

Wichtig: Die Notation in der Eingabedatei für `auto_alg` ist abhängig von der Groß- und Kleinschreibung. Ebenso muss darauf geachtet werden, dass Leerzeichen korrekt (wie im Beispiel dargestellt) gesetzt sind.

Nach der Deklaration des Problemtyps folgen in der Eingabedatei nun mehrere Blöcke, die mit `<CONTROLPARAMETERS> ... </CONTROLPARAMETERS>` umschlossen sind. Der erste dieser Blöcke hat eine besondere Bedeutung, hier werden die Parameter für den Zufallsgenerator und die Anzahl der Algorithmen angegeben. Parameter, die nicht benötigt werden, können weggelassen werden. Diese einzelnen Parameter haben folgende Bedeutung:

Parameter	**Bedeutung**
`M`, `N`	Gibt die Problemgröße $n \times m$ an, wobei n die Anzahl der Jobs und m die Anzahl der Maschinen ist.
`NUMBERPROBLEMS`	Gibt die Anzahl der Probleminstanzen an, die `auto_alg` generieren und lösen soll.
`NUMBERALGORITHMS`	Gibt die Anzahl der Algorithmen an, die auf eine Probleminstanz angewendet werden sollen.
`MINPT`, `MAXPT`	Minimale und maximale Werte für einen Eintrag in der Matrix der Bearbeitungszeiten.
`MINDD`, `MAXDD`	Minimale und maximale Werte für die Fälligkeitstermine der Jobs.
`MINRI`, `MAXRI`	Minimale und maximale Werte für die Bereitstellungszeiten der Jobs. Diese Parameter werden nur benötigt, wenn `r_i` im Problemtyp gesetzt wurde.
`MINWI`, `MAXWI`	Minimale und maximale Werte für die Wichtungsfaktoren der Jobs (wird für die Zielfunktionen $\sum w_i C_i$, $\sum w_i U_i$ und $\sum w_i T_i$ benötigt).
`TIMESEED`	Startzufallszahl für die Erzeugugn der Bearbeitungszeiten.
`MACHSEED`	Startzufallszahl für die Erzeugung der technologischen Reihenfolgen.
`DDSEED`	Startzufallszahl für die Erzeugung der Fälligkeitstermine der Jobs.
`RISEED`	Startzufallszahl für die Erzeugung der Bereitstellungszeiten der Jobs.
`WISEED`	Startzufallszahl für die Erzeugung der Wichtungsfaktoren der Jobs.
`RIDIMODE`	Wird dieser Parameter auf `1` gesetzt, werden die Fälligkeitstermine d_i nach folgender Formel berechnet: $$d_i = r_i + \mathrm{TF} \cdot \sum_{j=1}^{m} p_{ij}, \qquad i = 1, 2, \ldots n$$ Die Bereitstellungszeiten r_i der Jobs werden gleichmäßig im Intervall $[0, r_{\max}]$ verteilt. $$r_{\max} = \frac{1}{2n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}$$
`TFACTOR`	Setzt den Parameter TF (tightness factor) für den Fall `RIDIMODE` $= 1$.

Nach der Deklaration der Parameter für den Zufallsgenerator folgen nun weitere Parameterblöcke, die die nacheinander abzuarbeitenden Algorithmen darstellen. Hiervon muss es genau so viele geben, wie im Parameter `NUMBERALGORITHMS` angegeben wurde. Der erste Algorithmus sollte immer `lower_bounds` sein, um später Probleme mit der Filterung der Ausgabedaten zu vermeiden. Dieses Programm berechnet untere Schranken für C_{max} sowie $\sum C_i$ und berechnet als drittes Ergebnis den Erwartungswert von $\sum C_i$ unter der Voraussetzung, dass keine Stillstandszeiten existieren.

Die genaue Notation für den Aufruf eines speziellen Algorithmus wird im Kapitel **Algorithmen in LiSA** ($\rightarrow$ 4) beschrieben.

`auto_alg` arbeitet die Eingabedatei wie folgt ab:

- Generierung einer Probleminstanz aus dem Zufallsgenerator nach den angegebenen Parametern.
- Anwendung der Algorithmen auf die generierte Probleminstanz in der Reihenfolge, wie sie in der Eingabedatei angegeben sind. Gefundene Lösungen eines Algorithmus können dabei von einem der nachfolgenden Algorithmen als Grundlage genommen und iterativ verbessert werden.
- Abspeichern der Probleminstanz mit allen Lösungen, die von den Algorithmen gefunden wurden.

Diese Schritte werden so oft wiederholt, wie im Parameter `NUMBERPROBLEMS` angegeben wurde.

Extras zum Algorithmenaufruf

Der Aufruf `AUTOALG_START_FROM {ZAHL1,ZAHL2,...}` bestimmt, welche Ausgaben als Eingaben für den aktuellen Algorithmus verwendet werden sollen. `ZAHL1` und `ZAHL2` sind dabei die Nummern von bereits abgearbeiteten Verfahren. `{0}` bedeutet keine Eingabe für diesen Algorithmus. Der Standardwert für einen Algorithmus mit Nummer `i` ist `{i-1}`.
Durch den Algorithmus `best_of` wird die beste Lösung zu einer Instanz unter den angegebenen Verfahren ausgewählt. Dadurch können z.B. für alle Probleminstanzen die jeweils besten Lösungen verschiedener konstruktiver Verfahren als Startlösung eines iterativen Verfahrens dienen. Dieser Auswahlalgorithmus funktioniert wie ein selbständiges Verfahren und muss bei der Anzahl der Algorithmen im Dateikopf mitgezählt werden.

Im folgenden Beispiel soll als Startlösung für den nächsten Algorithmus die beste Lösung aus den von den Algorithmen 2, 5 und 10 erzeugten Lösungen dienen:

```
<CONTROLPARAMETERS>
string AUTOALG_EXECUTABLE best_of
string AUTOALG_START_from {2,5,10}
</CONTROLPARAMETERS>
```

Durch den Aufruf `AUTOALG_TIMELIMIT` kann für jeden Algorithmus ein Zeitlimit angegeben werden. Dann wird das Verfahren mit `SIGINT` unterbrochen, sobald das Zeitlimit abgelaufen

ist. Die dabei gemessene Zeit ist keine reine Prozesszeit, sondern reale Zeit. Der Standardwert ist `0` und entspricht keinem Limit. Dieser Parameter darf nur für Algorithmen benutzt werden, die eine Ausgabe erzeugen, wenn sie mit `SIGINT` abgebrochen werden.

Aufruf von Autoalg

Das Programm `auto_alg` befindet sich im LiSA-Hauptverzeichnis im Unterordner `bin`. Es muss genau aus diesem Ordner aufgerufen werden, da es von hier aus die abzuarbeitenden Algorithmen starten muss. Die erwartete Eingabedatei kann natürlich in einem beliebigen Verzeichnis liegen. In dem Verzeichnis, in dem die Eingabedatei liegt, legt `auto_alg` alle während seiner Abarbeitung gelösten Probleme ab. Diese können in LiSA geöffnet und angesehen werden.
`auto_alg` gibt während der Abarbeitung Logmeldungen auf die Standardausgabe aus. Diese Meldungen sind jedoch sehr zahlreich und unübersichtlich. Es empfiehlt sich also, diese von der Standardausgabe in eine Datei umzuleiten. Die ausgegebenen Informationen lassen sich dann besser auswerten, indem man Filter auf diese Datei anwendet.

Ein Aufruf von `auto_alg` kann unter Linux/Unix/Cygwin beispielsweise so aussehen:

```
cd ~/LiSA/bin
./auto_alg ~/LiSA-Daten/algorithmen.alg > ~/LiSA-Daten/auto_alg.out
```

Unter Windows (mit der Standard-Installation des LiSA-Paketes) kann die Eingabeaufforderung (Start/Ausführen → “cmd”) genutzt werden. Der entsprechende Befehl kann dort so aussehen:

```
cd C:\Programme\LiSA\bin
auto_alg.exe C:\LiSA-Daten\algorithmen.alg > C:\LiSA-Daten\auto_alg.out
```

Dies ruft `auto_alg` mit der Eingabedatei `algorithmen.alg` auf und leitet die Ausgabe in die Datei `auto_alg.out` um.

Auswertung der Resultate durch Filter

Die Filter liegen ebenfalls im Unterordner `bin`. Um sie auszuführen, muss Perl installiert sein. Beim Aufruf muss ihnen die vorher erzeugte Ausgabedatei von `auto_alg` übergeben werden. Die Filter schreiben ihre Ausgaben wiederum auf die Standardausgabe, diese lässt sich aber ebenfalls mit dem >-Operator in eine Datei umleiten.
Folgender Konsolenbefehl unter Linux/Unix/Cygwin ruft einen Filter auf die vorher erzeugte Ausgabedatei auf. Das Ergebnis der Filterung wird in die Datei `filter.out` geschrieben.

```
./filter_objectives ~/LiSA-Daten/auto_alg.out > ~/LiSA-Daten/filter.out
```

In der Windows-Eingabeaufforderung sieht der Befehl z.B. folgendermaßen aus:

```
perl filter_objectives C:\LiSA-Daten\auto_alg.out > C:\LiSA-Daten\filter.out
```

Anmerkung: Für den Aufruf der Filter über die Windows-Eingabeaufforderung ist darauf zu achten, dass Perl korrekt in der Umgebungsvariablen `PATH` eingetragen ist. Dies passiert bei der Installation von Perl aber meist automatisch.

Alle Filter listen ihre Ergebnisse in Tabellen auf. Jede Zeile enthält Informationen zu einer bearbeiteten Probleminstanz. Es wird immer davon ausgegangen, dass als erster Algorithmus `lower_bounds` ausgeführt wird. Die Ergebnisse des jeweils ersten Algorithmus werden dementsprechend von den Filtern nicht ausgegeben. Eine Ausnahme stellt `filter_runtime` dar, der auch die Laufzeit des ersten Algorithmus ausgibt.

filter	**Bedeutung**
`filter_runtime`	Listet spaltenweise für jeden Algorithmus die Laufzeit in Sekunden auf.
`filter_objectives`	Gibt in den ersten drei Spalten die Ergebnisse von `lower_bounds` für das Problem an. Die folgenden Spalten listen für jeden ausgeführten Algorithmus den erreichten Zielfunktionswert auf.
`filter_lmax`	Gibt in den ersten drei Spalten die Ergebnisse von `lower_bounds` für das Problem an. In den folgenden Spalten wird für jeden ausgeführten Algorithmus die erreichte maximale Verspätung der Jobs ausgegeben, auch wenn dies nicht die Zielfunktion war.
`filter_abort`	Gibt für jeden Algorithmus den Abarbeitungsstatus in Prozent an, falls gegeben. Dieser beträgt normalerweise 99 oder 100%, wenn der Algorithmus normal durchgelaufen ist. Einige Algorithmen haben jedoch Abbruchkriterien (eine Schranke für den Zielfunktionswert wurde erreicht, oder es wurde über zu viele Nachbarlösungen iteriert, ohne dass eine Verbesserung auftrat), sodass sie auch außerplanmäßig früher terminieren. Bei sehr schnellen Algorithmen wird "n/a" ausgegeben.
`filter_abortStucks`	Gibt für iterative Algorithmen an, ob sie vorzeitig terminiert sind. Dies kann passieren, wenn sich der Zielfunktionswert über lange Zeit nicht verbessert hat. Eine "1" bedeutet, der Algorithmus ist vorzeitig terminiert, bei "0" hat der Algorithmus die angegebene Anzahl an Iterationsschritten vollständig ausgeführt. Bei sehr schnellen Algorithmen steht hier ebenfalls "n/a".
`filter_sumti`	Gibt für jeden Algorithmus den erreichten Wert für $\sum T_i$ an (auch wenn dies nicht die Zielfunktion war). Die ersten drei Spalten listen die Ergebnisse von `lower_bounds` für das Problem auf.

Noch einmal: ACHTUNG!

Die Arbeit mit dem automatisierten Algorithmenaufruf muss sehr sorgfältig erfolgen, da die Eingabedatei nicht intern auf Richtigkeit überprüft wird. Insbesondere hat der Nutzer darauf zu achten, ob der aufgerufene Algorithmus überhaupt für den betrachteten Problemtyp zur Verfügung steht und ob die Parameter für die Probleminstanzerzeugung und die Algorithmen richtig sind (Gross- und Kleinbuchstaben und Leerzeichen beachten!). Eine weitere Fehlerquelle liegt in der Angabe der Anzahl der auszuführenden Algorithmen. Bitte richtig zählen!

Kapitel 8

Beispiele

In diesem Kapitel wird beschrieben, wie man einen Algorithmus über die Kommandozeile aufrufen kann und wie man die grafische Oberfläche von LiSA nutzt. Im Verzeichnis **LiSA/data/sample** sind einige Beispieldateien enthalten, das sind Eingabedateien für LiSA und Eingabedateien zum automatisierten Algorithmenaufruf.

8.1 Aufruf eines Algorithmus über die Kommandozeile

In diesem Beispiel soll eine gegebene Probleminstanz ohne die Benutzung der grafischen Oberfläche von LiSA gelöst werden. Dies erfolgt durch die Codierung der Probleminstanz in XML und dem anschließenden Aufruf eines Lösungsalgorithmus direkt über die Kommandozeile. Dieses Vorgehen kann dazu verwendet werden, um das Lösen und Optimieren gegebener Shop-Schedulingprobleme zu automatisieren.

Gegeben sei eine Probleminstanz vom Typ $J \mid\mid \sum C_i$ mit 4 Maschinen und 3 Aufträgen. Weiterhin sind die Bearbeitungszeiten und die technologische Reihenfolge gegeben:

$$PT = \begin{bmatrix} 53 & 98 & 0 & 80 \\ 88 & 38 & 89 & 96 \\ 37 & 30 & 0 & 60 \end{bmatrix} \quad \begin{array}{lllllllll} A_1 & : & M_2 & \rightarrow & M_4 & \rightarrow & M_1 & & \\ A_2 & : & M_2 & \rightarrow & M_1 & \rightarrow & M_3 & \rightarrow & M_4 \\ A_3 & : & M_4 & \rightarrow & M_2 & \rightarrow & M_1 & & \end{array}$$

Für diese Probleminstanz soll mit Hilfe des Algorithmus ***Beam Search*** ($\rightarrow$ 4.2.3) ein Zeitplan gefunden werden.

Ergänzen der fehlenden Angaben

Für die Übergabe an einen Lösungsalgorithmus müssen die Operationenmenge und die technologische Reihenfolge in Matrixform angegeben werden. Aus der Matrix der Bearbeitungszeiten PT lässt sich die folgende Operationenmenge ableiten:

$$SIJ = I \times J \backslash \{(13), (33)\} \Longrightarrow \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Die Matrix MO ergibt sich wie folgt:

$$MO = \begin{bmatrix} 3 & 1 & . & 2 \\ 2 & 1 & 3 & 4 \\ 3 & 2 & . & 1 \end{bmatrix}$$

Erzeugen des `instance`-Dokumentes

Die Probleminstanz muss nun in ein XML-Dokument überführt werden, das dem Lösungsalgorithmus übergeben werden kann. Wichtig ist auch die Angabe der Aufrufparameter für den Lösungsalgorithmus, die – in einem `<controls>`-Element zusammengefasst – in dem Dokument mit enthalten sind. Die einzelnen Parameter bestehen jeweils aus einem Namen, dem Typ (`string`, `integer` oder `real`) und einem Wert. Welche Parameter ein Algorithmus genau verlangt, lässt sich im Kapitel 4 nachschlagen.

Hinweis: Für das korrekte Einlesen der XML-Datei ist auch die Reihenfolge der XML-Elemente wichtig. Beispielsweise muss das `<controls>`-Element *direkt nach* dem `<values>`-Element folgen. Der allgemeine Aufbau eines `instance`-Dokumentes ist im Anhang A beschrieben.

Für die oben gegebenen Daten wird also das folgende `instance`-Dokument erstellt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE instance PUBLIC "" "LiSA.dtd">
<instance xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
  <problem>
    <alpha env="J"/>
    <beta/>
    <gamma objective="Sum_Ci"/>
  </problem>
  <values m="4" n="3">
    <processing_times model="lisa_native">
      {
        {  53  98   0  80 }
        {  88  38  89  96 }
        {  37  30   0  60 }
      }
    </processing_times>
    <operation_set model="lisa_native">
      {
        {   1   1   0   1 }
        {   1   1   1   1 }
        {   1   1   0   1 }
      }
    </operation_set>
    <machine_order model="lisa_native">
      {
        {   3   1   0   2 }
        {   2   1   3   4 }
        {   3   2   0   1 }
      }
    </machine_order>
```

```
    </values>
    <controls>
      <parameter type="string" name="MODE" value="INSERT" />
      <parameter type="string" name="INS_ORDER" value="LPT" />
      <parameter type="string" name="INS_METHOD" value="INSERT1" />
      <parameter type="string" name="CRITERION" value="OBJECTIVE" />
      <parameter type="integer" name="k_BRANCHES" value="5" />
    </controls>
  </instance>
```

Aufruf des Algorithmus unter Windows

Es wird angenommen, dass die eben erstellte XML-Datei unter dem Namen `beispiel.xml` im Ordner `C:\Scheduling\` liegt. Der aufzurufende Algorithmus (in diesem Fall `beam.exe`) liegt im Unterordner `bin` des LiSA-Verzeichnisses (standardmäßig `C:\Programme\LiSA\bin`). Mit Hilfe der Windows- Eingabeaufforderung (`cmd`) muss in diesen Ordner gewechselt werden, um von hier aus den Algorithmus aufzurufen (der Aufruf von einem anderen Arbeitsverzeichnis aus ist leider nicht möglich).

```
cd C:\Programme\LiSA\bin
beam.exe C:\Schedulng\beispiel.xml C:\Scheduling\beispiel.out.xml
```

Dieser Aufruf erzeugt im Ordner `C:\Scheduling\` die Datei `beispiel.out.xml`, welche ein Dokument vom Typ `solution` ist. In ihr ist die Ausgabe des Algorithmus und damit der gesuchte Zeitplan enthalten.

Aufruf des Algorithmus unter UNIX und cygwin

Hierbei wird angenommen, dass die erstellte XML-Datei im Home-Verzeichnis unter dem Namen `beispiel.xml` gespeichert wurde. Der aufzurufende Algorithmus (in diesem Fall `beam`) liegt im Unterordner `bin` des LiSA-Verzeichnisses (standardmäßig `~/LiSA/bin`). Mit Hilfe einer Konsole kann das Programm in diesem Arbeitsverzeichnis aufgerufen werden (ein Aufruf aus einem anderen Verzeichnis ist leider nicht möglich).

```
cd ~/LiSA/bin
./beam ~/beispiel.xml ~/beispiel.out.xml
```

Dieser Aufruf erzeugt im Home-Verzeichnis die Datei `beispiel.out.xml`, welche ein Dokument vom Typ `solution` ist. In ihr ist die Ausgabe des Algorithmus und damit der gesuchte Zeitplan enthalten.

8.2 Beispiel zur LiSA-Nutzung

Der Nutzer betrachtet ein open-shop Problem mit $m = 4$ Maschinen und $n = 4$ Aufträgen und Makespanminimierung. Es liegen keine weiteren Nebenbedingungen vor. Wie ist in diesem Fall LiSA zu nutzen? Nach dem Start von LiSA wird unter ***Datei*** der Button ***Neu*** angeklickt. Das Problemtyp-Fenster öffnet sich und das Problem wird in der $\alpha \mid \beta \mid \gamma$ Beschreibung eingegeben, d.h. als Maschinenumgebung wird O und als Zielfunktion wird Cmax gewählt.

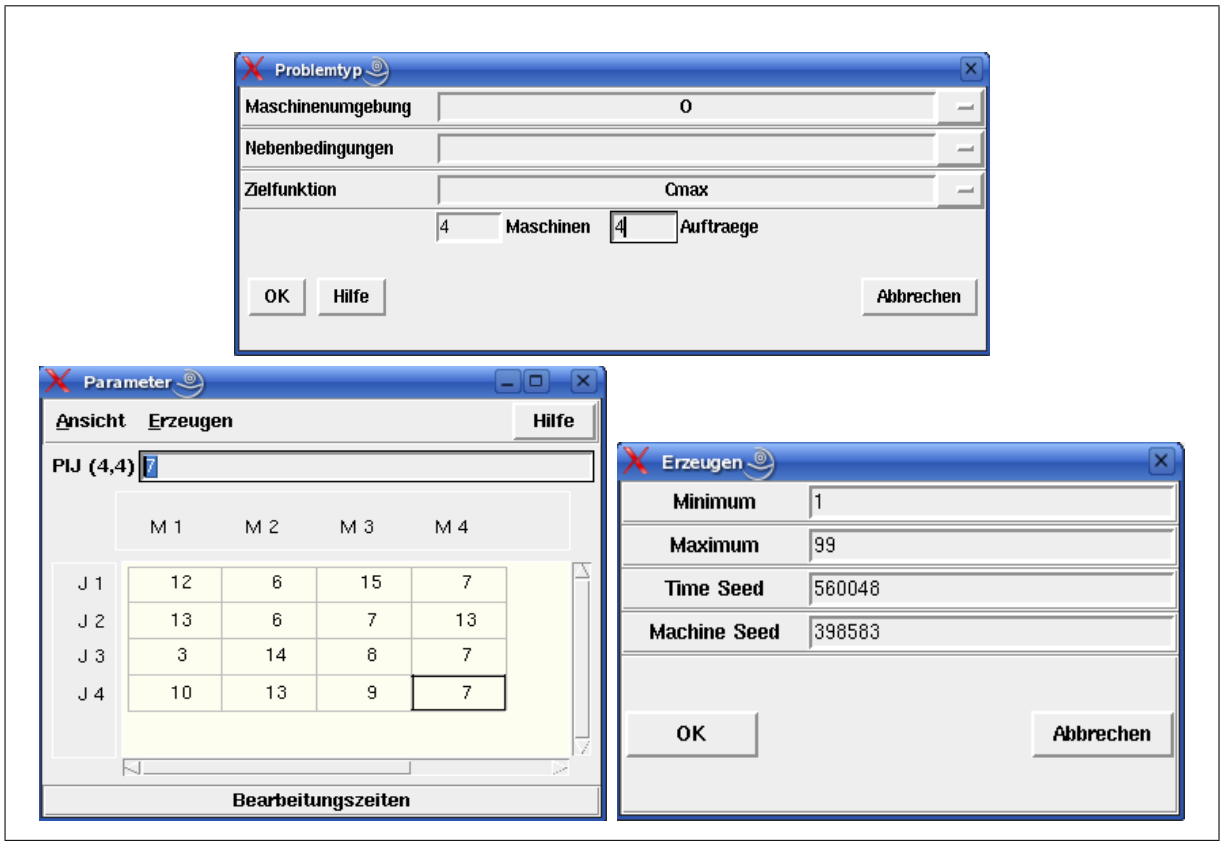


Abbildung 8.1: Eingabe einer Probleminstanz

Weiter wird sowohl die Anzahl der Maschinen als auch die Anzahl der Aufträge auf 4 gesetzt. Nun stellt LiSA alle Module zur Verfügung, die es für diesen Problemtyp enthält. Gestartet wird mit der Eingabe der Bearbeitungszeiten (Button ***Bearbeiten***, ***Parameter***, ***Erzeugen***, ***... der Bearbeitungszeiten***). Neben der Handeingabe, die für unser Beispiel benutzt wird, ist es möglich, einen Zufallszahlengenerator zu nutzen. Er erzeugt die Bearbeitungszeiten gleichverteilt aus dem Intervall [Minimum, Maximum]. Time Seed und Machine Seed sind beliebige Startparameter für den Generator, die bei gleicher Wahl gleiche Werte erzeugen. Die Abbildung 8.1 zeigt die entsprechenden LiSA Fenster. Die Matrix der Bearbeitungszeiten PT ist gegeben durch

$$PT = \begin{bmatrix} 12 & 6 & 15 & 7 \\ 13 & 6 & 7 & 13 \\ 3 & 14 & 8 & 7 \\ 10 & 13 & 9 & 7 \end{bmatrix}$$

Daneben können die Daten auch aus einer xml-Datei eingelesen werden, deren Format in Kapitel 2.5 beschrieben wird. Jetzt gibt LiSA die verfügbaren Algorithmen frei. Unter ***Algorithmen*** sind sowohl exakte als auch heuristische Algorithmen für das betrachtete Problem verfügbar.

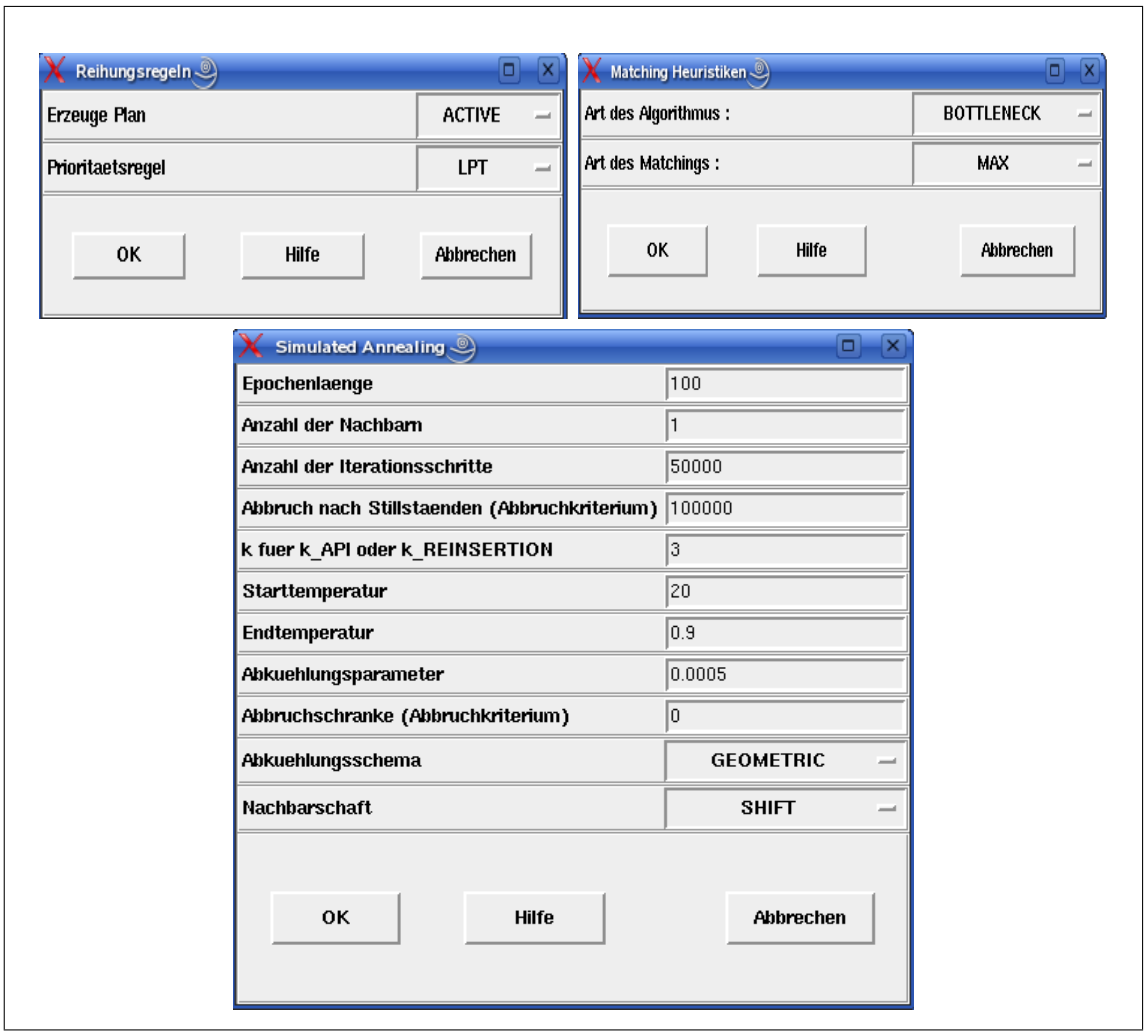


Abbildung 8.2: Heuristische Algorithmen

Die Abbildung 8.2 zeigt einige der verfügbaren Heuristiken. Schnelle Reihungsregeln, wie die LPT-Regel (longest processing time first), erzeugen einen ersten aktiven Schedule, mit dem verschiedene iterative Suchverfahren gestartet werden können. Hier wird das Fenster zu Simulated Annealing gezeigt. Verschiedene Parameter, wie z.B. die Nachbarschaft oder das Abkühlungsschema, können gewählt werden. Einzelheiten für die Parameter sind der Beschreibung des Suchverfahrens zu entnehmen. Ein weiteres konstruktives Verfahren ist die Anwendung eines Matchingalgorithmus, dabei werden Schritt für Schritt Operationen an den Plan angefügt, deren gleichzeitige Bearbeitung möglich ist, hier unter Minimierung der maximalen Bearbeitungszeit der gleichzeitig zu bearbeitenden Operationen.
LiSA stellt bei Anwendung eines Algorithmus sofort das Ganttdiagramm des erzeugten Schedules auf dem Bildschirm zur Verfügung. Alle Formen des Outputs sind in Abbildung 8.3 enthalten. So sind der Plan und die Matrix der Fertigstellungszeiten aller Operationen sowie

Abbildung 8.3: Ausgabe der Ergebnisse

ihre Visualisierungen im Plangraphen und im Ganttdiagramm unter ***Ansicht*** auswählbar. Ganttdiagramme können maschinen- und auftragsorientiert angezeigt werden, der kritische Weg kann hervorgehoben werden (Button ***Optionen***). Wenn die Anzahl der Maschinen oder der Jobs zu groß ist, d.h. das Ganttdiagramm zu komplex ist, kann die Zoomfunktion helfen. Durch eventuell mehrfaches Zoomen kann das Ganntdiagramm so vergrößert werden, dass alle Informationen beim Scrollen sichtbar sind.

LiSA hat einige Extras, zwei davon sind in Abbildung 8.4 enthalten.

Die Ganttdiagramme können manipuliert werden, d. h. eine Operation, die durch Klicken der rechten Maustaste ausgewählt wird, kann sowohl in der Technologie (MO) als auch in der Organisation (JO) um eine Position nach vorn oder hinten verschoben werden. Falls die Verschiebung zu einem azyklischen Plangraphen führt, wird eine Fehlermeldung ausgegeben. Daneben ist es möglich, diese Operation als Quelle oder Senke des Plangraphen zu wählen. Dadurch wird immer ein azyklischer Plangraph erzeugt.

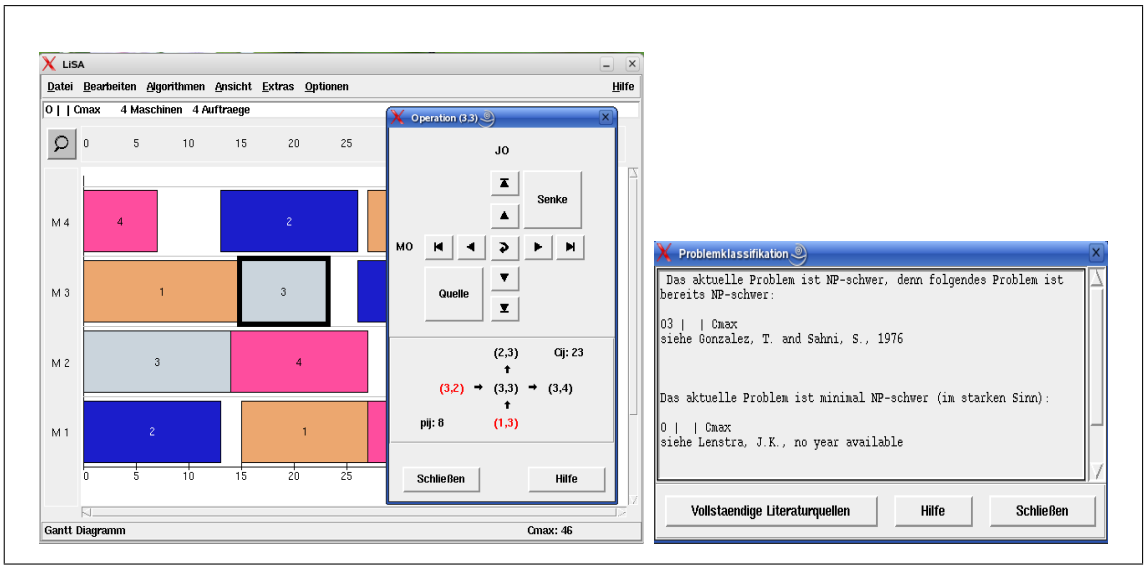


Abbildung 8.4: Manipulation und Komplexität

LiSA enthält als zweites Extra ein Komplexitätsmodul. Sobald in LiSA ein Problem in der $\alpha \mid \beta \mid \gamma$ Notation eingegeben wurde, kann die Komplexität des Problems unter ***Extras***, ***Problemklassifikation*** abgerufen werden. LiSA nutzt hier die Osnabrücker Datenbank zur Komplexität von deterministischen Schedulingproblemen. Darüber hinaus steht sogar die vollständige Literaturquelle zur Verfügung.

Literaturverzeichnis

[1] Adams,J., Balas, E., Zawack, D. [1988] *The Shifting Bottleneck Procedure for Job Shop Scheduling*; Management Science 34, 391-401

[2] Baker, K.R. [1984]: *Introduction to Sequencing and Scheduling*; Wiley & Sons, New York

[3] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J. [1996]: *Scheduling Computer and Manufacturing Processes*; Springer Verlag Berlin-Heidelberg-New York

[4] Blum,Ch. [2003]: *Beam-ACO-hybridizing ant colony optimization with beam search: an application to open shop scheduling*; computers & operations research III, Online verfügbar unter www.sciencedirect.com

[5] Bräsel, H.[1990]: *Latin Rectangle in Scheduling Theory*; Professorial Dissertation (in German), University Magdeburg, Germany

[6] Bräsel, H., Hennes, H.[2004]: *On the open-shop problem with preemption and minimizing the average completion time*; European Journal of Operational Research 157, 607-619

[7] Bräsel, H./Tautenhahn,T./Werner,F.[1993]: *Constructive Heuristic Algorithms for the Open Shop Problem*; Computing, 51, 95-110

[8] Brucker, P. [2001]: *Scheduling Algorithms*; Third Edition, Springer Verlag Berlin-Heidelberg-New York

[9] Chretienne, P., Coffman, E.G., Lenstra, J.K. (Editors) [1995]: *Scheduling Theory and its Applications*; John Wiley & Sons, Chichester-New York-Brisbane

[10] Conway, R.W., Maxwell, W.L., Miller, L.W. [1967]: *Theory of Scheduling*; Addison-Wesley Publishing Company, Massachusetts

[11] Dannenbring, D.G. [1977] *An Evaluation of Flowshop Sequencing Heuristics*; Managemant Science 23, 1174-1182

[12] Domschke, W., Scholl, A., Vo?, S. [1993]: *Produktionsplanung - Ablauforganisatorische Aspekte*; Springer Verlag Berlin-Heidelberg-New York

[13] French, S. [1982]: *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*; John Wiley & Sons, New York

[14] Gonzales, T., Sahni, S.[1976]: *Open-shop to minimize finish time*; Journal of the Association for Computing Machinery 23 (4), 665-679

[15] Graham, R.E., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. [1979]: *Optimization and approximation in deterministic sequencing and scheduling: a survey*; Ann. Discrete Math. 4, 287-326

[16] Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (Editors) [1993]: *Handbooks in Operations Research and Management Science (Volume 4): Logistics of Production and Inventory*; Elsevier Science Publishers B.V., North-Holland, Amsterdam-London-NewYork-Tokyo

[17] Lageweg, B.J., Lawlwer, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. [1981]: *Computer-aided Complexity Classification of Deterministic Scheduling Problems*; Report BM 138, Centre for Mathematics and Computer Science, Amsterdam

[18] Lenstra, J.K. [1977]: *Sequencing by enumerative methods*; Centre Tracts 69, Amsterdam

[19] Morton, T.E., Pentico, D.W. [1993]: *Heuristic Scheduling Systems*; John Wiley & Sons, Inc., New York

[20] Muth, J.F., Thompson, G.L. (Editors) [1963]: *Industrial Scheduling*; Prentice Hall, Englewood Cliffs

[21] Pinedo, M.: Scheduling [1995]: *Theory, Algorithms and Systems*; Prentice Hall, Inc., Englewood Cliffs, New Jersey

[22] Rinnooy Kan, A.H.G. [1976]: *Machine scheduling problems: Classification, Complexity and Computations*; Martinus Nijhoff/ The Hague,

[23] Tanaev, V.S., Sotskov, Y.N., Strusevich, V.A. [1994]: *Scheduling Theory: Multi-Stage Systems*; Kluwer Academic Publishers, Dordrecht

[24] Taillard, E. [1993]: *Benchmarks for basic scheduling problems*; European Journal of Operational Research 64, 278-285

[25] Tanaev, V.S., Sotskov, Y.N., Strusevich, V.A. [1994]: *Scheduling Theory: Multi-Stage Systems*; Kluwer Academic Publishers, Dordrecht

[26] Willenius, P. [2000] *Irreduzibilitẗstheorie bei Shop-Schedulingproblemen*; Shaker Verlag, Aachen, Dissertation

Anhang A

XML-Referenz

Dieses Kapitel soll einen detaillierten Einblick in den Aufbau sämtlicher XML-Dateien geben, die in LiSA verwendet werden. Es ergänzt den Abschnitt **Das File Format in LiSA** (→ 2.5), in dem die fünf Dokumenttypen vorgestellt wurden.

A.1 Allgemeines

A.1.1 Die Sprache XML

XML (*extensible markup language*) ist eine HTML-ähnliche Auszeichnungssprache für Informationen in Klartextform. Genau wie in HTML bestehen XML-Dokumente aus Elementen, die miteinander verschachtelt werden können, so dass eine Baumstruktur entsteht. Die Elemente können Text, Attribute und weitere Elemente enthalten. An dieser Stelle soll auf einige Begriffe eingegangen werden, die in dieser Dokumentation in Verbindung mit XML häufig benutzt werden. Diese Beschreibungen sind jedoch stark vereinfacht und dienen nur dem Grundverständnis der hier gezeigten Beispiele. Eine genauere Beschreibung der Sprache XML kann auf der Website des World Wide Web Consortium (www.w3.org/xml) eingesehen werden.

- Ein *Tag* ist eine in spitzen Klammern (`< >`) eingeschlossene Zeichenkette. Beginnt diese Zeichenkette mit einem Slash (`/`) spricht man von einem *Endtag*, andernfalls von einem *Starttag*. Das erste Wort der Zeichenkette ist der Name des Tags. Einem Starttag muss immer ein Endtag gleichen Namens folgen.
  Bsp. `<controls>  </controls>`

- Starttags können *Attribute* enthalten. Diese werden in der Form `[attribut]="[wert]"` mit einem Leerzeichen vom Tagnamen getrennt notiert.
  Bsp. `<schedule m="5" n="3" semiactive="yes">`

- Ein *Element* bezeichnet ein Paar von Start- und Endtags, zusammen mit dem von ihnen eingeschlossenen Text. Dieser Text kann wiederum selbst Elemente enthalten, die *Kindelemente* genannt werden.
  Bsp. `<due_dates model="lisa_native"> { 5 8 10 4 2 } </due_dates>`

- Besitzt ein Element keinen eingeschlossenen Text oder Kindelemente, kann die Kurzschreibweise `<[tagname] [attribut1]="[wert1]" ... />` benutzt werden, wodurch das Endtag entfällt.

A.1.2 Darstellung von Daten

Einfache Datentypen, wie ganze oder gebrochene Zahlen sowie Zeichenketten, werden einfach mit Hilfe von Attributen dargestellt. Ein Beispiel dafür ist das `<schedule>`-Tag:

```
<schedule m="10" n="20" semiactive="yes">
```

Hier wird die Problemgröße durch die Ganzzahlen $m = 10$ und $n = 20$ festgelegt. Zusätzlich besagt die Zeichenkette `"yes"` als Wert des Attributes `semiactive`, dass der folgende Zeitplan semiaktiv ist. Zu beachten ist bei reellen Zahlen die Verwendung des Dezimalpunktes anstatt eines Kommas.

Weiterhin werden Vektoren und Matrizen als komplexe Datentypen verwendet. Ein Vektor der Dimension n mit den Komponenten `a_1` bis `a_n` wird dargestellt durch:

```
{ a_1  a_2  a_3  ...  a_n }
```

Als Trennung zwischen den Werten können beliebig viele Whitespace-Zeichen benutzt werden. Eine $n \times m$ Matrix kann als n-dimensionaler Vektor dargestellt werden, der selbst m-dimensionale Vektoren enthält:

```
{
  { a_11  a_12  a_13  ...  a_1m }
  { a_21  a_22  a_23  ...  a_2m }
  ...
  { a_n1  a_n2  a_n3  ...  a_nm }
}
```

A.2 Die Dokumenttypen `problem`, `instance` und `solution`

Diese drei Dokumenttypen dienen allgemein zur Kommunikation von Schedulingproblemen in LiSA. Sie bauen Schrittweise aufeinander auf. So speichert ein `problem`-Dokument nur einen Problemtyp in $\alpha \mid \beta \mid \gamma$ – Notation. Ein `instance`-Dokument speichert zusätzlich eine konkrete Probleminstanz und ein `solution`-Dokument eine Probleminstanz zusammen mit einer oder mehreren Lösungen.

A.2.1 Aufbau

Stellvertretend für die drei Dokumenttypen ist hier der Aufbau eines `solution`-Dokumentes dargestellt. In `instance`-Dokumenten fehlen im Gegensatz dazu nur die `<schedule>`-Elemente, `problem`-Dokumente bestehen nur aus dem `<problem>`-Element.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE solution PUBLIC "" "LiSA.dtd">
<solution xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
  <problem>
    <alpha ... />
    <beta ... />
    <gamma ... />
  </problem>
  <values m="..." n="...">
    <processing_times model="lisa_native">
      ...
    </processing_times>
    <operation_set model="lisa_native">
      ...
    </operation_set>
    <due_dates model="lisa_native">
      ...
    </due_dates>
    <release_times model="lisa_native">
      ...
    </release_times>
    <weights model="lisa_native">
      ...
    </weights>
    <weights_2 model="lisa_native">
      ...
    </weights_2>
    <extra model="lisa_native">
      ...
    </extra>
  </values>
  <controls>
    <parameter type="..." name="..." value="..." />
    ...
  </controls>
  <schedule m="..." n="..." semiactive="...">
    <plan model="lisa_native">
      ...
    </plan>
    <machine_sequences model="lisa_native">
      ...
    </machine_sequences>
    <job_sequences model="lisa_native">
      ...
    </job_sequences>
    <completion_times model="lisa_native">
      ...
    </completion_times>
  </schedule>
  ...
```

```
</solution>
```

A.2.2 Beschreibung

<solution>

Dieses Element enthält eine oder mehrere Lösungen für eine Instanz eines Scheduling-Problems.
Elternelement: keines (Wurzelelement des Dokumenttyps `solution`)
Kindelemente: `<problem>`, `<values>`, `<schedule>`, `<controls>` (*optional*)

Dieses Element besitzt keine Attribute.

<instance>

Dieses Element enthält eine Instanz eines Scheduling-Problems.
Elternelement: keines (Wurzelelement des Dokumenttyps `instance`)
Kindelemente: `<problem>`, `<values>`, `<controls>` (*optional*)

Dieses Element besitzt keine Attribute.

<problem>

Dieses Element stellt einen Problemtyp dar. Es enthält je einen `<alpha>`, `<beta>` und `<gamma>`-Tag, die den Problemtyp in der gängigen $\alpha \mid \beta \mid \gamma$ – Notation darstellen.
Elternelement:

- keines im Dokumenttyp `problem` (Wurzelelement)
- `<instance>` im Dokumenttyp `instance`
- `<solution>` im Dokumenttyp `solution`

Kindelemente: `<alpha>`, `<beta>`, `<gamma>`

Dieses Element besitzt keine Attribute.

<alpha>

α beschreibt die Maschinenumgebung.
Elternelement: <problem>
Kindelemente: keine

Attribut	Bedeutung
env	Die Maschinenumgebung des Problems. Mögliche Werte sind 1, O, F, J, G, P, Q, R
m	(*optional*) Gibt die Anzahl der Maschinen an. Wird dieser Parameter weggelassen, ist die Maschinenanzahl variabel und gehört zur Problemgröße.

<beta>

β beschreibt die Jobcharakteristika und Nebenbedingungen.
Elternelement: <problem>
Kindelemente: keine

Attribut	Bedeutung
pmtn	(*optional*) Gibt an, ob Preemption erlaubt ist. Mögliche Werte sind yes, no.
prec	(*optional*) Gibt an, ob es Vorrangbedingungen zwischen den Aufträgen gibt. Mögliche Werte sind yes, no, intree, outtree, tree, sp_graph, chains.
release_times	(*optional*) Gibt an, ob es Bereitstellungstermine für die Aufträge gibt. Mögliche Werte sind yes, no.
due_dates	(*optional*) Gibt an, ob es Fälligkeitstermine für die Aufträge gibt. Mögliche Werte sind yes, no.
processing_times	(*optional*) Gibt Auskunft über die Bearbeitungszeiten der Aufträge. Mögliche Werte sind arbitrary, constant, uniform.
no-wait	(*optional*) Gibt an, ob Wartezeiten zwischen Operationen des selben Auftrags verboten sind. Mögliche Werte sind yes und no.

<gamma>

γ beschreibt die Zielfunktion.
Elternelement: <problem>
Kindelemente: keine

Attribut	Bedeutung
objective	Die Zielfunktion des Problems. Mögliche Werte sind Cmax, Lmax, Sum_Ci, Sum_wiCi, Sum_Ui, Sum_wiUi, Sum_Ti, Sum_wiTi, irreg_1, irreg_2

<values>

Dieses Element fasst alle Informationen zusammen, die zu einer Probleminstanz gehören.
Elternelement:

- <instance> im Dokumenttyp instance
- <solution> im Dokumenttyp solution

Kindelemente: <processing_times>, <operation_set>, <machine_order> (*optional*), <release_times> (*optional*), <due_dates> (*optional*), <weights> (*optional*), <weights_2> (*optional*), <extra> (*optional*)

Attribut	Bedeutung
m	Die Anzahl der Maschinen des Problems.
n	Die Anzahl der Aufträge des Problems.

<processing_times>

Dieses Element enthält die Matrix P der Bearbeitungszeiten.
Elternelement: <values>
Kindelemente: Eine Matrix vom Format n $\times$ m mit den Bearbeitungszeiten.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<operation_set>

Dieses Element enthält die Operationenmenge als binäre Matrix. Eine Eins an Stelle (i, j) bedeutet, dass die Operation (ij) ausgeführt werden muss, also Auftrag A_i auf Maschine M_j bearbeitet werden muss. Umgekehrt bedeutet eine Null an Stelle (i, j), dass die Operation (ij) nicht ausgeführt werden muss. p_{ij} muss in diesem Fall ebenfalls Null sein.
Elternelement: <values>
Kindelemente: Eine binäre Matrix vom Format n × m , die die Operationenmenge repräsentiert.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<machine_order>

Dieses Element enthält die Matrix MO, die die technologische Reihenfolge vorgibt.
Elternelement: <values>
Kindelemente: Eine Matrix vom Format n × m, die die technologische Reihenfolge repräsentiert.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<release_times>

Dieses Element enthält die Bereitstellungszeiten der Aufträge als n-dimensionalen Vektor. Die i-te Komponente des Vektors enthält dabei die Bereitstellungszeit des Auftrags A_i.
Elternelement: <values>
Kindelemente: Ein Vektor mit n Einträgen.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<due_dates>

Dieses Element enthält die Fälligkeitstermine der Aufträge als n-dimensionalen Vektor. Die i-te Komponente des Vektors enthält dabei den Fälligkeitstermin des Auftrags A_i.
Elternelement: <values>
Kindelemente: Ein Vektor mit n Einträgen.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<weights>

Dieses Element enthält die Wichtungsfaktoren w_i der Aufträge A_i als n-dimensionalen Vektor.
Elternelement: <values>
Kindelemente: Ein Vektor mit n Einträgen.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<weights_2>

Elternelement: <values>
Kindelemente: Ein Vektor mit n Einträgen.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<extra>

Elternelement: <values>
Kindelemente:

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<controls>

Dieses Element fasst alle Parameter zusammen, die einem mit diesem Dokument als Eingabe aufgerufenem Algorithmus übergeben werden sollen.
Elternelement:

- <instance> im Dokumenttyp instance
- <solution> im Dokumenttyp solution

Kindelemente: <parameter>

Dieses Element enthält keine Attribute.

<parameter>

Dieses Element stellt einen Parameter zur Übergabe an einen Algorithmus dar.
Elternelement: <controls>
Kindelemente: keine

Attribut	Bedeutung
type	Der Datentyp des Parameters. Mögliche Werte sind string, integer, real
name	Der Name des Parameters. Eine Liste der Algorithmen und deren zugehöriger Parameter findet sich im Kapitel 4.
value	Der eigentliche Wert des Parameters.

<schedule>

Dieses Element umfasst eine Lösung einer Probleminstanz. Sie kann neben einem Plan auch die Matrix der Fertigstellungszeiten sowie die technologischen und organisatorischen Reihenfolgen enthalten.
Elternelement: <solution>
Kindelemente: <plan>, <machine_sequences> (*optional*), <job_sequences> (*optional*), <completion_times> (*optional*)

Attribut	Bedeutung
m	Die Anzahl der Maschinen des Problems.
n	Die Anzahl der Aufträge des Problems.
semiactive	(*optional*) Gibt an, ob ein Schedule semiaktiv ist.

<plan>

Dieses Element enthält die Matrix LR, die den Plan der Lösung enthält.
Elternelement: <schedule>
Kindelemente: Ein lateinisches Rechteck vom Format n × m, das den Plan der Lösung darstellt.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<machine_sequences>

Dieses Element enthält die Matrix MO, die die technologischen Reihenfolgen der Lösung repräsentiert.
Elternelement: <schedule>
Kindelemente: Eine Matrix vom Format n × m mit der technologischen Reihenfolge.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<job_sequences>

Dieses Element enthält die Matrix JO, die die organisatorischen Reihenfolgen der Lösung repräsentiert.
Elternelement: <schedule>
Kindelemente: Eine Matrix vom Format n × m mit der organisatorischen Reihenfolge.

Attribut	Bedeutung
model	Reserviert, muss den Wert lisa_native haben.

<completion_times>

Dieses Element enthält die Matrix C der Fertigstellungszeiten.
Elternelement: <schedule>
Kindelemente: Eine Matrix vom Format n × m mit den Fertigstellungszeiten der Operationen.

Attribut	Bedeutung
`model`	Reserviert, muss den Wert `lisa_native` haben.

A.3 Der Dokumenttyp `algorithm`

Dieser Dokumenttyp dient zur Einbindung der Algorithmen in LiSA. Näheres zu deren Verwendung findet sich im Abschnitt **Algorithmenmodule** (→ 7.1).

A.3.1 Aufbau

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE algorithm PUBLIC "" "LiSA.dtd">
<algorithm
  xmlns:LiSA="http://lisa.math.uni-magdeburg.de"
  name="..."
  type="..."
  call="..."
  code="external"
  help_file="algorithm/...">

  <exact>
    <problem>
        <alpha ... />
        <beta ... />
        <gamma ... />
    </problem>
    ...
  </exact>
  <heuristic>
    <problem>
        <alpha ... />
        <beta ... />
        <gamma ... />
    </problem>
    ...
  </heuristic>
  <alg_controls>
    <integer name="..." caption="..." default="..." />
    ...
    <real name="..." caption="..." default="..." />
    ...
    <choice>
        <item name="..." caption="..." />
        ...
    </choice>
```

```
      ...
      <fixed name="..." value="..." />
      ...
    </alg_controls>
  </algorithm>
```

Konkrete Beispiele von `algorithm`-Dokumenten sind im Unterordner `src/algorithm/sample` zu finden.

A.3.2 Beschreibung

`<algorithm>`

Dieses Element umfasst die gesamte Algorithmenbeschreibung.
Elternelement: keines (Wurzelelement des `algorithm`-Dokumentes).
Kindelemente: `<heuristic>`, `<exact>`, `<alg_controls>` (*optional*).

Attribut	Bedeutung
`name`	Name des Algorithmus, wie er in LiSA im Menü **Algorithmen** angezeigt wird.
`type`	Entweder `constructive` oder `iterative`. Iterative Algorithmen benötigen eine bereits vorher durch einen konstruktiven Algorithmus erstellte Lösung, um darauf aufzubauen.
`code`	Reserviert, hier gibt es nur die Möglichkeit `external`
`call`	Name des Programms, das den Algorithmus implementiert, *ohne* Dateierweiterung (z.B. `.exe`). Dieser stimmt normalerweise mit dem Namen des Ordners überein, in dem der Quellcode des Algorithmus liegt.
`help_file`	Name der HTML-Hilfedatei (*mit* Dateierweiterung, z.B. `.html`).

`<exact>` und `<heuristic>`

`<exact>` und `<heuristic>` verhalten sich sehr ähnlich. Sie enthalten Problembeschreibungen, wobei Probleme unter `<exact>` vom Algorithmus garantiert optimal gelöst werden, Probleme unter `<heuristic>` jedoch nur näherungsweise.
Elternelement: `<algorithm>`
Kindelemente: `<problem>`

Diese Elemente besitzen keine Attribute.

<problem>

Dieses Element stellt innerhalb einer <exact> oder <heuristic>-Umgebung einen lösbaren Problemtyp dar. Es enthält je einen <alpha>, <beta> und <gamma>-Tag, die den Problemtyp in der gängigen $\alpha \mid \beta \mid \gamma$ – Notation darstellen.

Elternelement: <heuristic> oder <exact>

Kindelemente: <alpha>, <beta>, <gamma>

Dieses Element besitzt keine Attribute.

<alpha>

α beschreibt die Maschinenumgebung.

Elternelement: <problem>

Kindelemente: keine

Attribut	Bedeutung
env	Die Maschinenumgebung des Problems. Mögliche Werte sind 1, O, F, J, G, P, Q, R
m	(*optional*) Gibt die Anzahl der Maschinen an. Wird dieser Parameter weggelassen, ist die Maschinenanzahl variabel und gehört zur Problemgröße.

<beta>

β beschreibt die Jobcharakteristika und Nebenbedingungen.

Elternelement: <problem>

Kindelemente: keine

Attribut	Bedeutung
pmtn	(*optional*) Gibt an, ob Preemption erlaubt ist. Mögliche Werte sind yes, no.
prec	(*optional*) Gibt an, ob es Vorrangbedingungen zwischen den Aufträgen gibt. Mögliche Werte sind yes, no, intree, outtree, tree, sp_graph, chains.
release_times	(*optional*) Gibt an, ob es Bereitstellungstermine für die Aufträge gibt. Mögliche Werte sind yes, no.
due_dates	(*optional*) Gibt an, ob es Fälligkeitstermine für die Aufträge gibt. Mögliche Werte sind yes, no.
processing_times	(*optional*) Gibt Auskunft über die Bearbeitungszeiten der Aufträge. Mögliche Werte sind arbitrary, constant, uniform.
no-wait	(*optional*) Gibt an, ob Wartezeiten zwischen Operationen des selben Auftrags verboten sind. Mögliche Werte sind yes und no.

<gamma>

γ beschreibt die Zielfunktion.
Elternelement: <problem>
Kindelemente: keine

Attribut	Bedeutung
objective	Die Zielfunktion des Problems. Mögliche Werte sind Cmax, Lmax, Sum_Ci, Sum_wiCi, Sum_Ui, Sum_wiUi, Sum_Ti, Sum_wiTi, irreg_1, irreg_2

<alg_controls>

Über das Element <alg_controls> können Parameter deklariert werden, die dem Algorithmus übergeben werden.
Elternelement: <problem>
Kindelemente: <integer> (*optional*), <real> (*optional*), <choice> (*optional*), <fixed> (*optional*)

Dieses Element enthält keine Attribute.

<integer>

Deklariert einen ganzzahligen Parameter, der dem Algorithmus übergeben werden kann.
Elternelement: <alg_controls>
Kindelemente: keine

Attribut	Bedeutung
name	Der Name des Parameters, mit dem er intern angesprochen wird.
caption	Der Name des Parameters, in einer einfachen Form. Dies ist die Bezeichnung, wie sie im Parametereingabefenster nach der Auswahl Algorithmus dargestellt wird.
default	Der Standardwert für diesen Parameter.

<real>

Deklariert einen reellwertigen Parameter, der dem Algorithmus übergeben werden kann.
Elternelement: <alg_controls>
Kindelemente: keine

Attribut	Bedeutung
name	Der Name des Parameters, mit dem er intern angesprochen wird.
caption	Der Name des Parameters, in einer einfachen Form. Dies ist die Bezeichnung, wie sie im Parametereingabefenster nach der Auswahl eines Algorithmus dargestellt wird.
default	Der Standardwert für diesen Parameter.

<choice>

<choice>-Parameter bieten eine Auswahl an verschiedenen Elementen.
Elternelement: <alg_controls>
Kindelemente: <item>

Attribut	Bedeutung
name	Der Name des Parameters, mit dem er intern angesprochen wird.
caption	Der Name des Parameters, in einer menschenlesbaren Form. Dies ist die Bezeichnung, wie sie im Parametereingabefenster nach der Auswahl eines Algorithmus dargestellt wird.

<item>

Items definieren die Werte, die ein <choice>-Parameter annehmen kann.
Elternelement: <choice>
Kindelemente: keine

Attribut	Bedeutung
name	Der Name des Wertes.

<fixed>

<fixed>-Parameter übergeben dem Algorithmenmodul feste Werte. Sie werden in der LiSA-Benutzeroberfläche nicht angezeigt und können nicht verändert werden; es kann also ein verstecktes Datum übergeben werden.
Elternelement: <alg_controls>
Kindelemente: keine

Attribut	Bedeutung
name	Die ID des Parameters.
value	Der Wert des Parameters. Dieser kann nicht vom Benutzer verändert werden.

A.4 Der Dokumenttyp controls

A.4.1 Aufbau

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE controls PUBLIC "" "LiSA.dtd">
<controls xmlns:LiSA="http://lisa.math.uni-magdeburg.de">
  <parameter type="integer" name="WIDTH" value="600"/>
  <parameter type="integer" name="HEIGHT" value="500"/>
  <parameter type="string" name="LANGUAGE" value="german"/>
  <parameter type="string" name="HTML_VIEWER" value="konqueror"/>
</controls>
```

A.4.2 Beschreibung

<controls>

Das <controls>-Element umfasst alle Parameterdefinitionen.

Elternelement: keines (Wurzelelement des `controls`-Dokumentes)
Kindelemente: `<parameter>`
Dieses Element besitzt keine Attribute.

<parameter>

Dieses Element stellt einen Programmparameter zur Übergabe an LiSA dar.
Elternelement: `<controls>`
Kindelemente: keine

Attribut	Bedeutung
`type`	Der Datentyp des Parameters. Mögliche Werte sind `string`, `integer`, `real`
`name`	Der Name des Parameters.
`value`	Der eigentliche Wert des Parameters.

A.4.3 LiSA-Programmparameter

Typ	Name	Bedeutung
`integer`	`WIDTH`	Fensterbreite von LiSA beim Programmstart (in Pixel).
`integer`	`HEIGHT`	Fensterhöhe von LiSA beim Programmstart (in Pixel).
`string`	`STARTFILE`	Dateiname eines Problems in XML-Format, das beim Programmstart von LiSA automatisch geladen werden soll.
`string`	`HTML_VIEWER`	Der Konsolenbefehl, den LiSA benutzen soll, um den standard-HTML-Browser zu starten, z.B. `iexplore` (Windows), `konqueror` oder `firefox` (Linux)
`string`	`LANGUAGE`	Hier kann die Programmsprache ausgewählt werden (entweder `english` oder `german`).

Anhang B

GNU-Lizenzbedingungen

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are refering to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so

that any problems introduced by others will not reflect on the original authors' reputations. Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.
The precise terms and conditions for copying, distribution and modification follow.

Terms and conditions for copying, distribution and modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Here in after, translation is included without limitation in the term "modification".) Each license is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transfering a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program

itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licenses extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, more aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the prefered form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system n on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a license cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation

excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. **Because the Program is licensed free of charge, there is no warranty for the Program, to the extent permitted by applicable law. except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.**

12. **In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.**

END OF TERMS AND CONDITIONS