

LiSA - A Library of Scheduling Algorithms

Version 2.3

Heidemarie Bräsel, Lars Dornheim, Sandra Kutz,
Marc Mörig, Ivo Rössling
Faculty of Mathematics
Otto-von-Guericke University Magdeburg

Abstract

This technical report reflects the current results in the development of the software package “LiSA - A Library of Scheduling Algorithms”. The scientific work of the research group was supported by the Ministry of Culture of the Land Sachsen-Anhalt in the projects “Latin Rectangles in Scheduling Theory” (October 1997 - September 1999) and “LiSA - A Library of Scheduling Algorithms” (November 1999 - October 2001). Within these projects four degree dissertations, three dissertations and one professorial dissertation were produced.

LiSA is a software package for solving deterministic scheduling problems. In the version 2.3 LiSA is now prepared for cooperative development. The report gives an overview on the obtained results. Each newcomer will find necessary information how to work with LiSA, each advancer can use this report as reference book.

The LiSA-team, the goals of LiSA, the license and the technical background of the software package are described. A short introduction into the used classification and models is given. An overview on the available algorithms is contained, which can be also used outside of LiSA. An example illustrates how LiSA is working. The LiSA file structure is explained and it is described in detail, how new algorithms can be inserted. Finally, the components of the cooperative development are explained.

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

1 The LiSA-Team

Working groups at universities are rare homogenously composed over a long time period. The following survey contains the names and task fields of all scientists and students who contributed in the development of LiSA. We start with the parents of LiSA:

- **Prof. Dr. Heidemarie Bräsel:** Initiator of the project, leader of the LiSA team and supervisor of the students and PhD students
- **PD Dr. Thomas Tautenhahn:** Responsible for the efficiency of data structures and of algorithms, supervisor of the students in programming (Oct.97 - Nov.00, professorial dissertation 2002)
- **Dr. Per Willenius:** Responsible for the development of the graphical user interface, the corresponding algorithms and for the error free connection of all program modules, supervisor of the students in programming (Oct.97 - Oct.01, dissertation 2000)
- **Dr. Martin Harborth:** Responsible for the complexity module in the LiSA main program, supervisor of the students in programming (Oct.97 - Sept.99, dissertation 1999)
- **Lars Dornheim:** Responsible for the compatibility of LiSA on different computer systems under different operating systems (Oct.98 - Aug.99)

In this first time period the following students implemented algorithms for LiSA:

- **Ines Wasmund:** Visualization of schedules in Gantt charts, evaluation module
- **Andreas Winkler:** Neighbourhood search strategies with different neighbourhood graphs
- **Marc Mörig:** Matching algorithms for open-shop problems, irreducibility tests
- **Christian Schulz:** Shifting-Bottleneck heuristic for the job shop problem
- **Manuela Vogel:** Heuristics for the flow-shop problem, composition of sequences

For a short time period **Holger Hennes, Birgit Grohe, Christian Tietjen, Carsten Malchau** and **Tanka Nath Dhamala** (Sandwich-scholar, Dissertation 2002) also worked in the project.

In the first computer lab (2001) in our faculty the following students implemented the modules:

- **Thomas Klemm:** Optimal rules for two machine shop problems
- **Andre Herms/Jan Tusch:** Beam-Search algorithms for the problem $O \parallel C_{max}$
- **Ivo Rössling:** Gonzales/Sahni algorithm for the problem $O | pmtn | C_{max}$
- **Marco Kleber:** Some dynamic dispatching rules
- **Claudia Isensee:** Pseudopolynomial algorithm for the $P2 \parallel C_{max}$ problem

In the last year **Lars Dornheim, Sandra Kutz, Marc Mörig** and **Ivo Rössling** prepared LiSA for cooperative development. The modularity of the software was increased and many bugs were corrected. A server was prepared for the communication between users and developers, a versions management system and a bug tracking system were installed. The concept of inserting own algorithms was updated and simplified. A new homepage was created. In short:

LiSA - Version 2.3 is finished!

This technical report reflects the obtained status of LiSA in the version 2.3.

2 What is LiSA?

LiSA is a software package for solving deterministic scheduling problems. In a scheduling problem a set of jobs should be processed on a set of machines under different additional constraints so that an objective function will be minimized. In the literature such scheduling problem is usually given by the triple $\alpha | \beta | \gamma$, where α describes the machine environment, β gives the additional constraints and γ represents the objective function.

The software LiSA has modular structure. LiSA's main part contains all basic algorithms connected with the model, the input and output procedures and the moduls of the graphical user interface. Additionally the program data will be managed here and the work with external algorithms will be coordinated. The complexity status of a problem in the notation $\alpha | \beta | \gamma$ can also be determined. All algorithms in LiSA used for solving scheduling problems are encapsulated in external modules. These modules are autonomous binaries with a common command line interface. They communicate with the main program via files. So they can be used both from within the LiSA GUI and independently without any GUI in batch mode. The insertion of new algorithm in LiSA works through an algorithm description file and a corresponding help file in html format. The used denotations, the

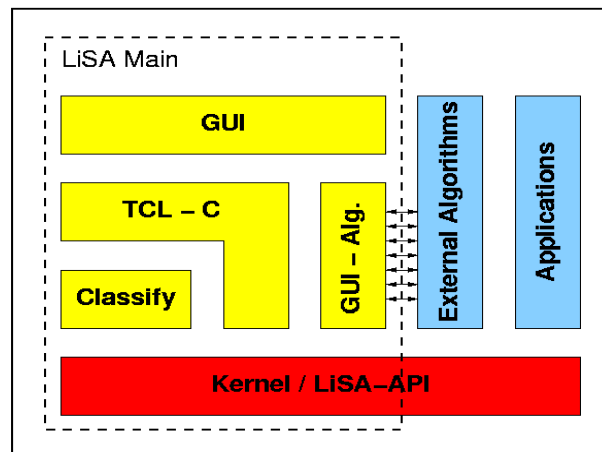


Figure 1: Structure of LiSA

classification and the models used in LiSA are given in Section 4. Section 5 contains an overview on the already implemented algorithms. In Section 6 an example illustrates how to work with LiSA. The LiSA file format for the input and output of an instance and the solution are described in Section 7. More information on the module interface and a detailed description how to insert own algorithms are contained in the sections 8 and 9. Finally, the new components on the LiSA server for the cooperative development are introduced in Section 10.

This report gives an overview on the software package without guarantee of completeness. "Learning by doing!" is the best way to use this software, this is assisted by the LiSA help.

3 License, Systems and Requirements

LiSA is licensed under the GPL (GNU General Public License), see Appendix.

LiSA is primarily created on and designed for Unix systems. To compile it, an ISO standard compliant C++ compiler and Tcl 7.5/Tk 4.1 are required. We recommend to use gcc 3.2 and Tcl/Tk 8.0 or higher, which are available for almost any Unix like systems. The current LiSA version was completely developed on SuSE Linux 8.1. Test builds were made on the following systems:

- *SuSE 8.0 using gcc 2.95.3 and Tcl/Tk 8.3;*
- *SuSE 8.1 using gcc 3.2 and Tcl/Tk 8.4;*
- *SuSE 8.1 using icc 7.0 and Tcl/Tk 8.4;*
- *Solaris 8 using gcc 2.95.3 and Tcl 7.6/Tk 4.2.*

Previous LiSA versions could successfully be compiled on:

- *SunOS 5.6/5.7 (sparcSTATION and ULTRAsparc);*
- *IRIX 6.4;*
- *HP-UX 09.05/10.10/10.20;*
- *AIX 4.2;*
- *SuSE 6.1;*
- *RedHat 5.1.*

Windows builds are made using Cygwin, which provides a Unix like environment for Windows. Precompiled Windows packages only are provided so that you don't need to install Cygwin yourself. So, our Windows builds require no further software to be installed. The current package was successfully installed and run on WindowsXP and Windows 98, but should work on any other Windows system, too.

4 Basic Knowledge

This is a short introduction into the notations and definitions in the area of deterministic scheduling problems that are used in LiSA. We start with some basic notations, explain sequences and schedules and describe the $\alpha|\beta|\gamma$ classification for such problems. To understand the input data and the output data of LiSA the block matrices model and the connection to the well-known disjunctive graph model are given. Furthermore, some fundamental algorithms based on these models are described.

4.1 Basic notations

In a *scheduling problem* a set of *jobs* has to be processed on a set of *machines* in a certain *machine environment* under certain *additional constraints* such that an *objective function* is optimized. The problem is called deterministic if all parameters are fixed and given in advance. Various optimization problems of allocation of restricted resources can also be modelled as scheduling problems. Table 1 contains some basic notations used in LiSA.

In a scheduling problem with more than one machine we introduce:

Basic notations	
n	number of jobs
m	number of machines
$\{A_1, \dots, A_n\}$	set of jobs which have to be processed
$I = \{1, \dots, n\}$	set of indices of the jobs
$\{M_1, \dots, M_m\}$	set of machines which process the set of jobs
$J = \{1, \dots, m\}$	set of indices of the machines
$p_{ij} \geq 0$	processing time of job A_i on machine M_j
$PT = [p_{ij}]$	matrix of processing times
(ij)	operation, i.e. the processing of job A_i on machine M_j , provided that $p_{ij} > 0$
SIJ	set of all operations (ij)
u_i	number of operations of job A_i
v_j	number of operations on machine M_j
c_{ij}	completion time of the operation (ij)
$C = [c_{ij}]$	matrix of completion times
r_i	release date of job A_i
d_i	due date of job A_i

Table 1: Basic notations for deterministic scheduling problems

The *machine order of the job A_i* is the order of machines on which this job has to be processed: $M_{j_1}^i \rightarrow M_{j_2}^i \rightarrow \dots \rightarrow M_{j_{u_i}}^i$, where j_1, \dots, j_{u_i} is a permutation of $1, \dots, u_i$.

The *job order on machine M_j* is the order of the jobs which this machine processes: $A_{i_1}^j \rightarrow A_{i_2}^j \rightarrow \dots \rightarrow A_{i_{v_j}}^j$, where i_1, \dots, i_{v_j} is a permutation of $1, \dots, v_j$.

If it is not confusing we will write these orders without superscripts.

4.2 Classification of deterministic scheduling problems

In LiSA the $\alpha \mid \beta \mid \gamma$ classification of deterministic scheduling problems developed by GRAHAM ET AL. [12] is used, where

- α describes the machine environment,
- β gives the job characteristics and additional constraints and
- γ is the objective function.

Note, that in LiSA this classification is not only applied for the description of the input problem but also for determining the complexity status of the considered problem and to prepare the menus of exact and heuristic algorithms that are implemented in the software package.

The Tables 2, 3 and 5 give an overview about possible parameters for the $\alpha \mid \beta \mid \gamma$ classification without the guarantee of completeness. In the literature it is sometimes allowed for job shop problems that a job can be processed several times on the same machine. In LiSA this is not the case.

Machine environment $\alpha = \alpha_1\alpha_2$	
$\alpha_1 \in \{1, P, Q, R\}$	Each job consists of exactly one operation which can be processed on any of the given machines.
$\alpha_1 = 1$	There is only one machine available, therefore $p_{i1} = p_i$ holds.
$\alpha_1 = P$	Each job is processed on exact one of m identical parallel machines, i.e. $p_{ij} = p_i$ is valid.
$\alpha_1 = Q$	Each of the m parallel machines has the same <i>speed</i> s_j , i.e. $p_{ij} = p_i/s_j$ holds.
$\alpha_1 = R$:	The speeds s_{ij} for the processing of an operation (ij) depends on both the job A_i and the machine M_j , therefore $p_{ij} = p_i/s_{ij}$ holds.
$\alpha_1 \in \{O, F, J\}$	Each job is processed on each machine exactly once or at most once (classical case).
$\alpha_1 = O$	open shop problem: The machine orders and the job orders can be chosen arbitrarily.
$\alpha_1 = J$	job shop problem: The machine orders are fixed and the job orders have to be determined.
$\alpha_1 = F$	flow shop problem: The machine orders are fixed and identical for each job, w.l.o.g.: $M_1 \rightarrow M_2 \rightarrow \dots \rightarrow M_m$. The job orders have to be determined.
$\alpha_2 \in \{\circ, c\}$	Characterization of the number of machines
$\alpha_2 = c$	The number of machines m is constant, $m = c$.
$\alpha_2 = \circ$	The number of machines is variable, i.e. a part of the problem size input.

Table 2: Parameters of the machine environment α

Characteristics of the jobs and additional constraints $\{\beta_1, \dots, \beta_5\}$	
$\beta_1 \in \{\circ, pmtn\}$	Preemption of operations
$\beta_1 = \circ$	Preemption is not allowed.
$\beta_1 = pmtn$	Preemption is allowed, i.e. the processing of a job on a machine can be interrupted and continued later on.
$\beta_2 \in \{\circ, prec, outtree, intree, tree, chain\}$	The precedence constraints $A_i \rightarrow A_k$ means, job A_i must be completed before job A_k can start.
$\beta_2 = \circ$	There are no precedence constraints.
$\beta_2 = chain$	The precedence constraints form a path.
$\beta_2 = outtree$	Each job has at most one predecessor.
$\beta_2 = intree$	Each job has at most one successor.
$\beta_2 = tree$	The precedence constraints are represented by a tree.
$\beta_2 = prec$	The precedence constraints form an acyclic digraph.
$\beta_3 \in \{\circ, r_i \geq 0, \}$	Release dates of the jobs
$\beta_3 = \circ$	Each job is available at time 0: $r_i = 0 \quad \forall i \in I$.
$\beta_3 = r_i \geq 0$	For each job A_i a release date $r_i \geq 0$ is given.
$\beta_4 \in \{\circ, d_i\}$	Due dates of the jobs
$\beta_4 = \circ$	There is no strictly due date for a job demanded to keep.
$\beta_4 = d_i$	Each job has to be completed before its due date $d_i \geq 0$.
$\beta_5 \in \{\circ, p_{ij} = 1\}$	Special processing times
$\beta_5 = \circ$	$p_{ij} \geq 0 \quad \forall (i, j) \in I \times J, p_{ij}$: integers.
$\beta_5 = p_{ij} = 1$ for $\alpha_1 \in \{1, O, F, J\}$	$p_{ij} = 1$ holds for all operations $(ij) \in SIJ$.

Table 3: Some additional constraints contained in β

Note, that β contains no, one or some parameters of the set $\{\beta_1, \dots, \beta_5\}$. Here additional conditions are possible, for instance the following:

- *no-wait* : Waiting times are not allowed between two operations of the same job.
- *no-idle* : Idle times between two operations on the same machine are not allowed.
- $p_{ij} \in \{1, 2\}$: Only processing times which are equal to 1 or to 2 are allowed.

To describe the objective function we introduce the following notations:

Further notations	
w_i	weight of job A_i
C_i	completion time of job A_i
$L_i = C_i - d_i$	lateness of job A_i
$T_i = \max\{0, C_i - d_i\}$;	tardiness of job A_i
$U_i = \begin{cases} 0, & \text{if } C_i \leq d_i \\ 1, & \text{otherwise} \end{cases}$	unit penalty of job A_i

Table 4: Characteristics of job J_i

Each of the objective functions $F(C_1, \dots, C_n)$ given in Table 5 is *regular*, i.e. if $C_i^* \geq C_i \forall i \in I$ holds then $F(C_1^*, \dots, C_n^*) \geq F(C_1, \dots, C_n)$ is satisfied.

A nonregular objective function is for instance the earliness-tardiness function, where both earliness and tardiness generate penalty costs dependent on $C_i - d_i$. These costs have to be minimized.

Objective functions $\gamma \in \{f_{max}, \sum f_i\}$	
$f_{max} \in \{C_{max}, L_{max}\}$	Objective function: $f_{max} \rightarrow \min!$
$C_{max} = \max_{i \in I} \{C_i\} \rightarrow \min!$	Minimize the makespan!
$L_{max} = \max_{i \in I} \{L_i\} \rightarrow \min!$	Minimize the maximum lateness!
$\sum f_i \in \{\sum C_i, \sum T_i, \sum U_i, \sum w_i C_i, \sum w_i T_i, \sum w_i U_i\}$	Objective function: $\sum f_i \rightarrow \min!$
$\sum C_i \rightarrow \min!$	Minimize the total completion time!
$\sum T_i \rightarrow \min!$	Minimize the total tardiness!
$\sum U_i \rightarrow \min!$	Minimize the number of tardy jobs!
$\sum w_i C_i \rightarrow \min!$	Minimize the total weighted completion time!
$\sum w_i T_i \rightarrow \min!$	Minimize the total weighted tardiness!
$\sum w_i U_i \rightarrow \min!$	Minimize the weighted number of tardy jobs!

Table 5: Regular objective functions

There may be further conditions for the processing. For instance, a flexible flow shop problem (*FFS*) is a combination of flow shop and parallel shop:

$$\left\{ \begin{array}{c} M_1^1 \\ M_2^1 \\ \vdots \\ M_{i_1}^1 \end{array} \right\} \rightarrow \left\{ \begin{array}{c} M_1^2 \\ M_2^2 \\ \vdots \\ M_{i_2}^2 \end{array} \right\} \rightarrow \dots \rightarrow \left\{ \begin{array}{c} M_1^s \\ M_2^s \\ \vdots \\ M_{i_s}^s \end{array} \right\}$$

A mixed shop problem is a combination of job shop and open shop.

Furthermore, there exist scheduling problems with resource constraints, batching problems, etc ...

4.3 Models for Shop Problems

To understand the input data and the output data of LiSA the used models will be explained briefly. We consider shop problems and we assume that each job is processed on at most one machine at a time and each machine processes at most one job at a time.

4.3.1 Sequences and Schedules

We define the following graphs where in each case the set of vertices is the set of operations SIJ :

- The graph of machine orders $G(MO)$ contains all arcs which describe the direct precedence constraints in all machine orders.
- The graph of job orders $G(JO)$ contains all arcs which describe the direct precedence constraints in all job orders.
- The graph $G(MO, JO) = (SIJ, A)$ contains all arcs which are contained in $G(MO)$ and $G(JO)$, i.e.

$$((ij), (kl)) \in A \iff \left\{ \begin{array}{l} (i = k \wedge \text{after the processing of job } A_i \text{ on} \\ M_j \text{ this job is processed on machine } M_l) \vee \\ (j = l \wedge \text{after the processing on machine } M_j \text{ of job } A_i \\ \text{this machine processes the job } A_k.) \end{array} \right.$$

A combination (MO, JO) of machine orders and job orders is called *feasible*, if the corresponding digraph $G(MO, JO)$ does not contain any cycle. In this case $G(MO, JO)$ is named a *sequence graph*.

Example 1 Three jobs have to be processed on four machines. The matrix PT of processing times is given by

$$PT = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 3 & 4 & 3 \\ 1 & 5 & 1 & 2 \end{bmatrix}, \text{ therefore } SIJ = I \times J \setminus \{(13)\} \text{ holds.}$$

We consider the following machine orders and job orders:

$$\begin{array}{ll}
A_1 : M_1 \rightarrow M_2 \rightarrow M_4 & M_1 : A_1 \rightarrow A_2 \rightarrow A_3 \\
A_2 : M_2 \rightarrow M_4 \rightarrow M_1 \rightarrow M_3 & M_2 : A_2 \rightarrow A_3 \rightarrow A_1 \\
A_3 : M_4 \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 & M_3 : A_3 \rightarrow A_2 \\
& M_4 : A_3 \rightarrow A_1 \rightarrow A_2
\end{array}$$

The corresponding graph $G(MO, JO)$ contains vertical arcs, which represent the job orders on the machines and horizontal arcs with respect to the machine orders of the jobs.

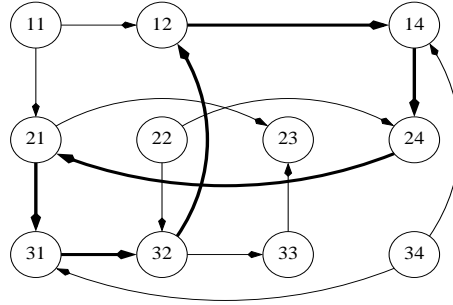


Figure 2: $G(MO, JO)$ for Example 1

The combination of machine orders and job orders is not feasible because $G(MO, JO)$ contains the cycle $(1, 2) \rightarrow (1, 4) \rightarrow (2, 4) \rightarrow (2, 1) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (1, 2)$. Each operation of this cycle is a predecessor and a successor from itself. Of course, there can not exist any time table of processing.

If we choose in each machine order and job order the natural order of the jobs and the machines, respectively, the graph $G(MO, JO)$ cannot contain any cycle, because all arcs are directed to the left or downwards. In this case a corresponding time table can easily be constructed.

The digraphs $G(MO)$ and $G(JO)$ are obtained from $G(MO, JO)$ by deleting the vertical arcs or the horizontal arcs, respectively.

Now we weight each vertex (ij) of the sequence graph $G(MO, JO)$ with the processing time p_{ij} . Then a time table of the processing is denoted as *schedule*. Usually schedules are described by the start times or the completion times of all operations. There exist the following classes of schedules:

A schedule is called *non-delay*, if no machine is kept idle when there exists a job available for processing.

A schedule is called *active*, if no operation can be completed earlier by changing the job orders without delaying any other operation.

A schedule is called *semiactive*, if no operation can be completed earlier without changing the job order on any of the machines.

Note, that each non-delay schedule is also active and each active schedule is also semiactive, but the converse does not hold.

In the case of regular objective functions it can be proved, that there always exists an optimal semiactive schedule. Therefore, we can use the calculation of a critical path in $G(MO, JO)$

to determine the completion time of all jobs, called *makespan*. A critical path in a directed acyclic graph is a path with maximal sum of all weights of the vertices which are contained in the path.

In LiSA semiactive schedules are described by the matrix $C = [c_{ij}]$ of completion times c_{ij} of all operations $(ij) \in SIJ$. Schedules are visualized by *Gantt charts*, which can be *machine oriented* or *job oriented*.

4.3.2 The Block Matrices Model

In the literature the most commonly used model for shop problems is the well-known disjunctive graph model, see for instance BRUCKER [5]. The model applied in LiSA can be derived from this by the following considerations:

- Cut the inserted source and sink and the corresponding incident arcs.
- Determine an acyclic orientation of the disjunctive graph.

If we allow in this graph only the direct precedence constraints in the machine orders and in the job orders, we obtain the sequence graph, used in LiSA.

The block matrices model contains a whole block of matrices, in each matrix we find in row i and column j an information on the operation (ij) , this is the real advantage of the model. After using the matrices sometimes you will also see the corresponding graphs and the structure of the contained paths without drawing the graphs.

Let an acyclic digraph be given. The *rank* $rk(v)$ of a vertex is defined as the number of vertices on a path with maximal number of vertices from a source to the vertex v . By means of this definition we describe $G(MO)$, $G(JO)$ and the sequence graph $G(MO, JO)$ by the following matrices:

- The *machine order matrix* $MO = [mo_{ij}]$: mo_{ij} is the rank of the operation $(ij) \in SIJ$ in the graph $G(MO)$.
- The *job order matrix* $JO = [jo_{ij}]$: jo_{ij} is the rank of the operation $(ij) \in SIJ$ in the graph $G(MO)$.
- The *sequence (matrix)* $LR = [lr_{ij}]$: lr_{ij} is the rank of the operation $(ij) \in SIJ$ in the sequence graph $G(MO, JO)$.

The block matrices model is illustrated in Figure 3. Note, that in this section also the matrices $H = [h_{ij}]$ and $T = [t_{ij}]$ are introduced which represent the completion time of all predecessors and of all successors of each operation $(ij) \in SIJ$, respectively.

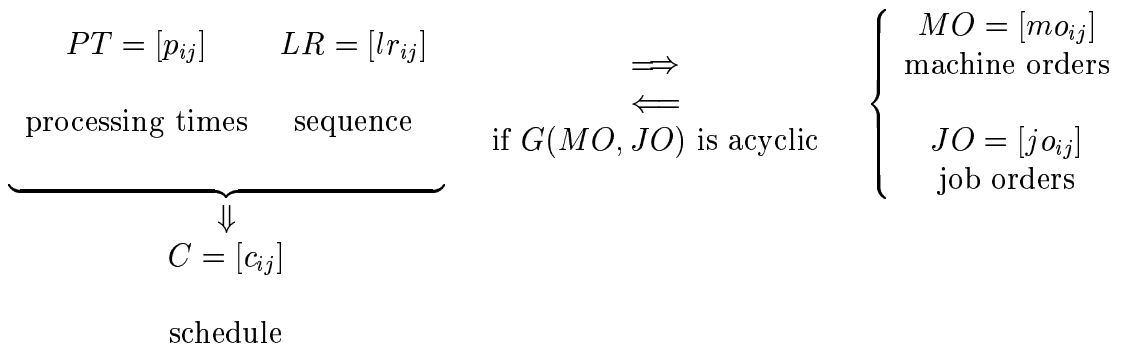


Figure 3: Block matrices model

Algorithm 1 determines the sequence matrix by means of the corresponding pair of MO and JO , if the combination (MO, JO) is feasible. The set MQ contains all operations, which are sources in both $G(MO)$ and $G(JO)$.

**Algorithm 1: Determination of LR by means of MO and JO ,
if $G(MO, JO)$ is acyclic**

Input: n, m, SIJ, MO and JO on the operation set SIJ ;
Output: LR on the operation set SIJ , if $G(MO, JO)$ is acyclic;
BEGIN $k := 0$;
 REPEAT
 $k := k + 1$; Calculate the set $MQ = \{(ij) \in SIJ \mid mo_{ij} = jo_{ij} = 1\}$;
 IF $MQ = \emptyset$ **THEN** (MO, JO) is infeasible and **STOP**;
 FORALL $(ij) \in MQ$ **DO**
 BEGIN
 $lr_{ij} = k$; Mark in MO the row i and in JO the column j ;
 END;
 $SIJ := SIJ \setminus MQ$;
 FORALL $(ij) \in SIJ$ in a marked row in MO **DO** $mo_{ij} := mo_{ij} - 1$;
 FORALL $(ij) \in SIJ$ in a marked column in JO **DO** $jo_{ij} := jo_{ij} - 1$;
 UNTIL $SIJ = \emptyset$;
END.

Algorithm 2 calculates MO and JO by means of LR . Here a_i and b_j are the smallest integers, which are available for the rank of an operation of job A_i and of an operation on machine M_j , respectively. The maximal entry in the sequence is denoted by r in the following algorithms.

Algorithm 2: Determination of MO and JO by LR

Input: n, m, r, I, J, SIJ, LR on the operation set SIJ ;
Output: MO und JO on the operation set SIJ ;
BEGIN Set $\forall i \in I: a_i = 1$ and $\forall j \in J: b_j = 1$;
 FOR $k := 1$ **TO** r **DO**
 FORALL $(ij) \in SIJ$ with $lr_{ij} = k$ **DO**
 BEGIN
 Set $mo_{ij} = a_i$ and $a_i = a_i + 1$;
 Set $jo_{ij} = b_j$ and $b_j = b_j + 1$;
 END;
 END.

Algorithm 3 generates the semiactive schedule, i.e. the matrix of completion times of all operations, by means of a corresponding pair PT and LR . Here r_i and \bar{r}_j denote the current smallest possible start time of operation (ij) , i.e. for job A_i and on machine M_j , respectively.

Algorithm 3: Determination of C by means of PT und LR **Input:** n, m, r, I, J, SIJ, PT and LR on the operation set SIJ ;**Output:** C on the operation set SIJ .**BEGIN**Set $\forall i \in I: r_i = 0$ and $\forall j \in J: \bar{r}_j = 0$;**FOR** $k := 1$ **TO** r **DO****FORALL** $(ij) \in SIJ$ mit $lr_{ij} = k$ **DO****BEGIN** $c_{ij} := \max\{r_i, \bar{r}_j\} + p_{ij}$; $r_i := c_{ij}$; $\bar{r}_j := c_{ij}$;**END**;**END.**

Algorithm 4 determines the matrices $H = [h_{ij}]$ and $T = [t_{ij}]$. h_{ij} is the head of operation (ij) , i.e. the weight of a maximal weighted path from a source to the operation (ij) . t_{ij} denotes the tail of operation (ij) , i.e. the weight of a maximal weighted path from operation (ij) to a sink in the sequence graph. Here r_i, \bar{r}_j is again the earliest start time for job A_i and on machine M_j , respectively. s_i, \bar{s}_j denote the earliest start time of job J_i and on machine M_j in the backwards calculation.

Algorithm 4: Determination of H and T **Input:** n, m, r, I, J, SIJ, PT and LR on the operation set SIJ ;**Output:** H and T on the operation set SIJ .**BEGIN**Set $\forall i \in I: r_i = 0$ and $\forall j \in J: \bar{r}_j = 0$;Set $\forall i \in I: s_i = 0$ and $\forall j \in J: \bar{s}_j = 0$;**FOR** $k := 1$ **TO** r **DO****BEGIN****FORALL** $(ij) \in SIJ$ with $lr_{ij} = k$ **DO****BEGIN** $h_{ij} := \max\{r_i, \bar{r}_j\}$; $r_i := h_{ij} + p_{ij}$; $\bar{r}_j := h_{ij} + p_{ij}$;**END**;**FORALL** $(ij) \in SIJ$ with $lr_{ij} = r - k + 1$ **DO****BEGIN** $t_{ij} := \max\{s_i, \bar{s}_j\}$; $s_i := t_{ij} + p_{ij}$; $\bar{s}_j := t_{ij} + p_{ij}$;**END**;**END**;**END.**

Note, that the matrix $W = H + PT + T$ contains for each operation the weight of a maximal weighted path from a source over the considered operation to a sink in the sequence graph. Therefore, all operations with maximal value in W belong to at least one critical path. Because of the structure of a sequence matrix we can sort all operations by nonincreasing

ranks in linear time. Therefore, all algorithms above described using this sorting have time complexity $O(nm)$.

This section is closed with an example to illustrate the block matrices model.

Example 2 Consider the matrix PT of processing times of Example 1, where the due dates $d_1 = 6, d_2 = 12, d_3 = 8$ are given. The following combination of machine orders and job orders is feasible because the corresponding graph $G(MO, JO)$ does not contain a cycle.

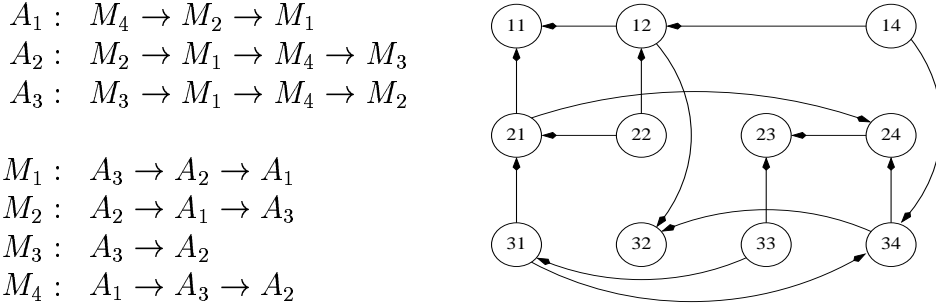


Figure 4: Machine orders, job orders and the sequence graph $G(MO, JO)$

Algorithm 1 determines the sequence LR and Algorithm 3 calculates the schedule C :

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

The matrices H and T are obtained by Algorithm 4, therefore $W = H + PT + T$ can be determined:

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

The schedule C yields $C_{max} = 12$ and $C_1 = 7, C_2 = 12, C_3 = 9$, therefore $\sum C_i = 28, L_{max} = 1, \sum T_i = 2$ and $\sum U_i = 1$ follow. Note, that this schedule is optimal in the open shop case for the objective functions C_{max} and L_{max} , there exist better schedules for $\sum C_i, \sum T_i$ and $\sum U_i$.

Note, that in the case of a job shop problem with given matrix MO all sequences LR are feasible that contains the machine orders of MO .

5 Algorithms in LiSA

Lisa provides a number of algorithms for various problem types. Use the help files to get more information. Most algorithms focus on shop problems.

5.1 All-purpose algorithms

- Exact algorithms

A general Branch-and-Bound algorithm for regular objective functions solves most open, job or flow shop problems. Our algorithm applies an insertion technique, see BRÄSEL [3] and BRÄSEL ET AL. [4], and user given upper bound and lower bounds. An upper bound can also be derived from previously applied heuristics. Given a good set of bounds, the algorithm can be sped up significantly.

- Constructive heuristic algorithms

Dispatching rules can be used for various objectives and several additional constraints. Step by step a new operation is inserted by means of a given strategy.

Rule	Strategy of choice of the next operation
RAND	random
FCFS	first come, first serve
EDD	earliest due date first
LQUE	with smallest difference of due date and (processing time + tail)
SPT	shortest processing time first
WSPT	weighted shortest processing time first
ECT	(reachable) earliest completion time first
WI	largest weight first
LPT	largest processing time first

Table 6: Dispatching rules

- Iterative algorithms

In LiSA several neighbourhood search algorithms are available. Each vertex of a neighbourhood graph corresponds to a solution (sequence and schedule) and is weighted with the objective function value of the considered problem. The set of arcs or edges is different in several neighbourhoods. We need a first solution, which can be found by a simple constructive heuristic. Then we start a walking tour (iterative search) on the neighbourhood graph to find a better solution.

Method	Description
Iterative improvement	moving to a better neighbouring solution - after enumeration of all neighbours or - if the first better schedule in the neighbourhood is found
Simulated annealing	allows the move to a worse solution with a certain probability, which is decreasing step by step
Threshold accepting	allows moving to a worse neighbour by means of a threshold, which is gradually reduced
Tabu search	generates tabu lists, to avoid to get back to solutions already visited

Table 7: Neighbourhoods search strategies

In Table 7 the different implemented iterative search methods are explained. To get more information we suggest the books of BRUCKER [5], BLAZEWICZ AT AL. [2] or PINEDO [18]. Table 8 contains the neighbourhoods available in LiSA.

Neighbourhood	Description
API	switches two (directly) neighbouring operations on a machine
SHIFT	shifts one operation arbitrarily on a machine
CR_API	switches two neighbouring operations on a machine, critical to C_{max} objective
3_CR	extends CR_API, exchanges predecessor and successor with other operations as well
BL_SHIFT	shifts a block of operations on a machine, critical to C_{max} objective
BL_API CR_SHIFT	switch operations on a machine in such way that one critical path in the schedule is destroyed (blockapproach idea)

Table 8: Neighbourhoods

5.2 Special algorithms

Some algorithms in LiSA were designed for special, well-defined problem classes. Table 9 contains exact algorithms and Table 10 gives an overview on heuristic algorithms.

Most algorithms contained in Table 9 are described in each book on scheduling theory. Note, that the original Branch and Bound algorithm of the group of P. Brucker is available in the Internet.

Problem type	Description
1 L_{max}	Branch and Bound
1 $\sum w_i T_i$	Smith rule, works like the WSPT rule
1 L_{max}	ERD rule, earliest release date first
F2 C_{max}	Johnson's rule
J2 C_{max}	Jackson's rule
O2 C_{max}	Gonzales/ Sahni algorithm
P2 C_{max}	Pseudopolynomial algorithm (no visualisation yet)
J C_{max}	Original Brucker's Branch-and-Bound Algorithm
O pmtn C_{max}	Gonzales/ Sahni (no visualisation yet)
O2 C_{max}	LAPT rule, longest alternating processing time - processes the job with the longest processing time on the other machine first

Table 9: Exact algorithms for special problems

Problem type	Description	Literature
F C_{max}	Dannenbring's flow shop heuristic	[8]
J C_{max}	Shifting bottleneck heuristic	[11]
O C_{max} and O $\sum C_i$	Matching heuristics	[4]
O C_{max} and F C_{max}	Beam search	[4]

Table 10: Heuristics for special problems

In version 2.3 the following problems are still unsolved:

- scheduling problems with precedence constraints,
- basic algorithms and visualisation of preemption problems,
- efficient data structures for and visualisation of parallel machine problems.

6 Example

The user would like to solve an open-shop problem with $m = 4$ machines, $n = 4$ jobs and makespan minimization without any additional constraint. The processing times are given by the following matrix PT :

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

What steps are necessary? After starting LiSA we choose under **File** the button **New**. The Problem Type window is opened and you enter your problem in the $\alpha | \beta | \gamma$ denotation, that means $O || C_{max}$. Don't forget to enter $n = 4$ and $m = 4$. Now LiSA provides us with all modules for the considered problem. We start with the input of the processing times (buttons **Edit**, **Parameter**, **Generate**, **Processing times**). You can do it by hand or by using the random generator. Figure 5 shows the corresponding LiSA windows. Note, that all input data can also read from a file, whose format is described in Section 7.

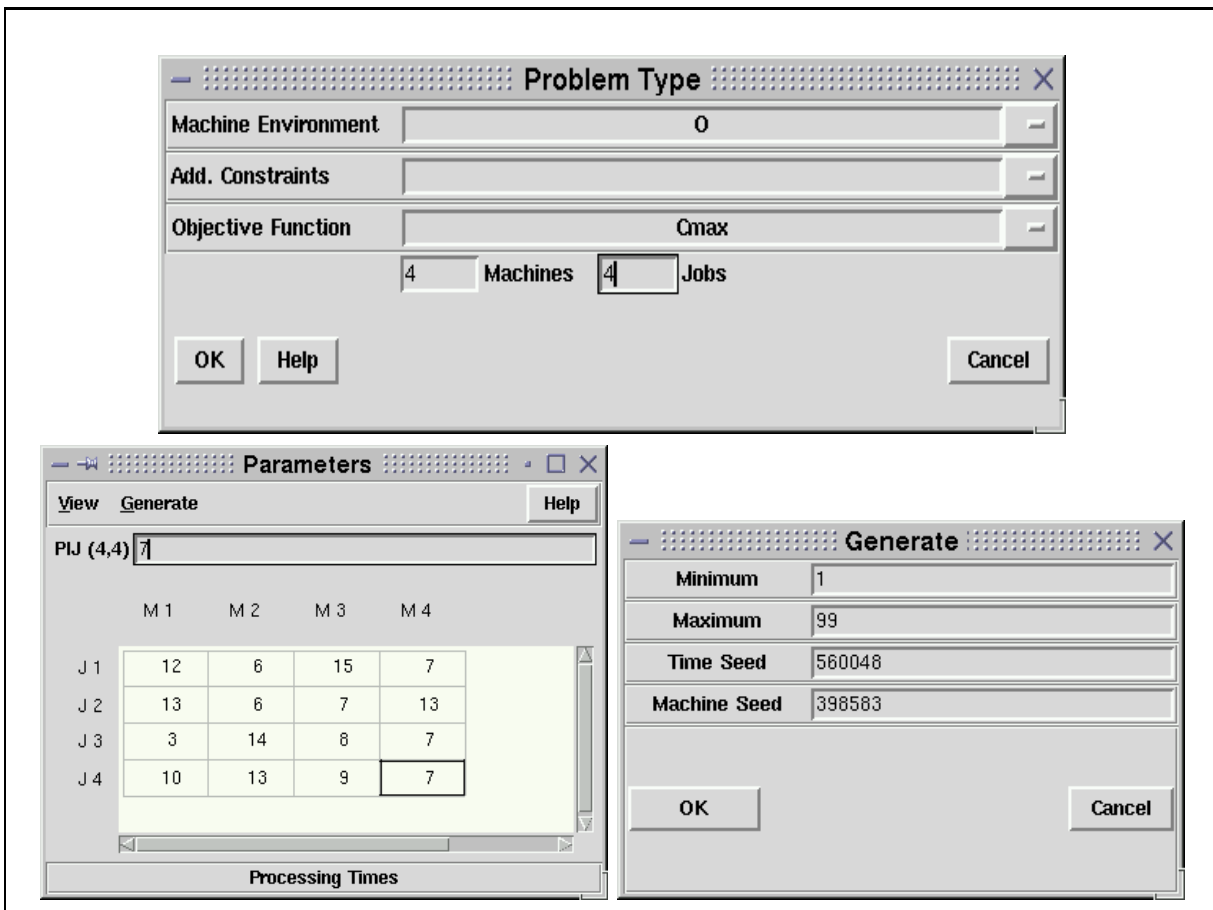


Figure 5: Input of the example

Under **Algorithms** exact and heuristic algorithms are available for the considered problem. Figure 6 shows some possibilities. For instance, the LPT-Rule (Longest Processing

Time First) can be applied. After the construction of a first schedule we are able to use all contained neighbourhood search algorithms, here simulated annealing with 3_CR neighbourhood is chosen. Some parameters can be varied, follow the description in LiSA. Finally, the matching heuristics are available for your problem: step by step all operations which belong to an optimal matching with respect to the processing time matrix can be processed simultaneously, here with minimal bottleneck objective function.

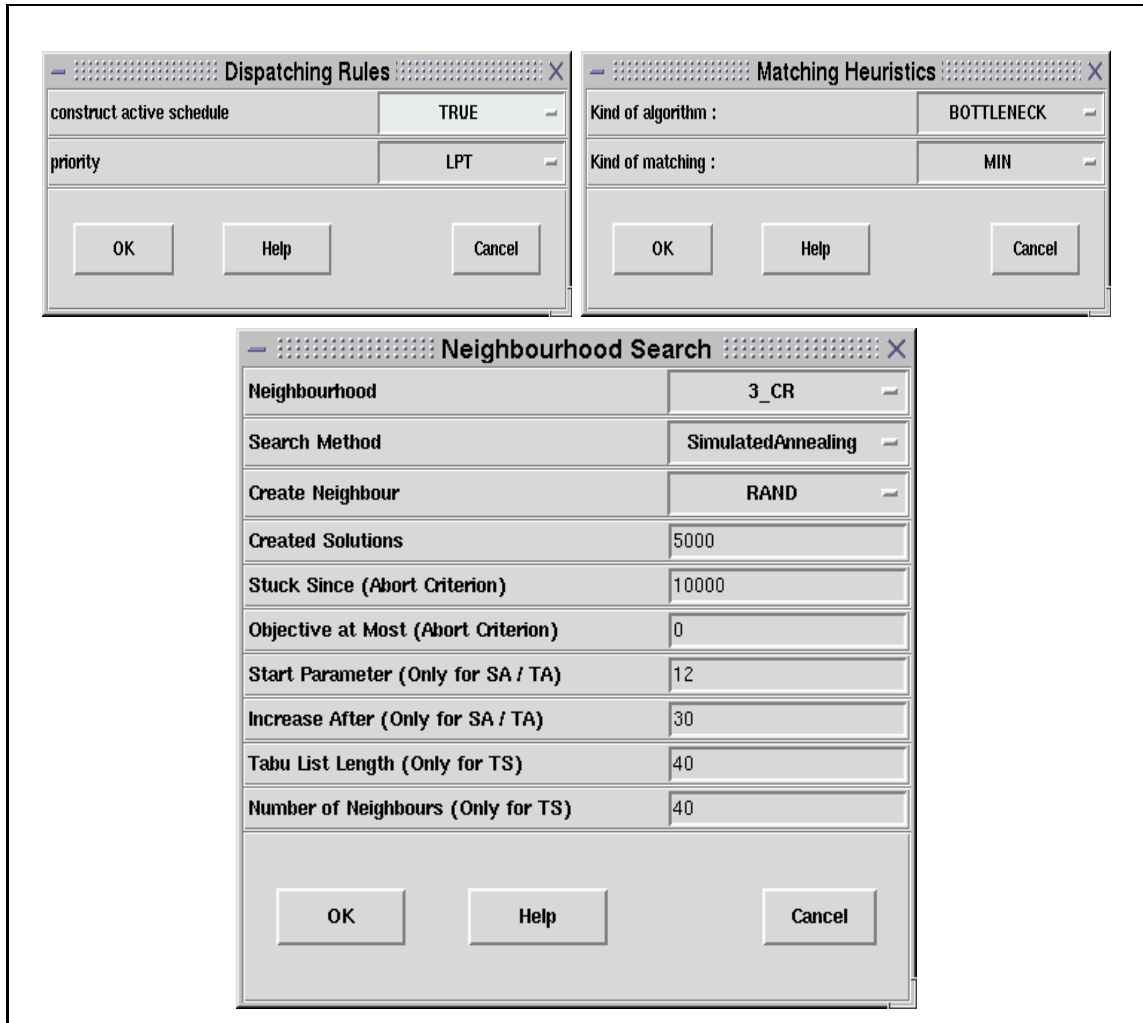


Figure 6: Heuristic algorithms for the considered problem

Usually the request of an algorithm produces the Gantt chart of the generated schedule, but there are also other output possibilities, see Figure 7. Here you can choose under **View** the sequence matrix or the sequence graph, the schedule described as matrix of completion times or as Gantt chart. Note, that under **Options** some options of the Gantt chart can be chosen, it can be for instance machine oriented or job oriented or the critical path may be highlighted. Moreover, some colours for the operations can be changed. If the number of jobs or machines is large, i.e. the Gantt chart is too complex, the use of the zoom makes sense.

LiSA has some extras, two of them are contained in Figure 8. The most important extra is the complexity modul. Whenever LiSA has the $\alpha | \beta | \gamma$ denotation of a problem it

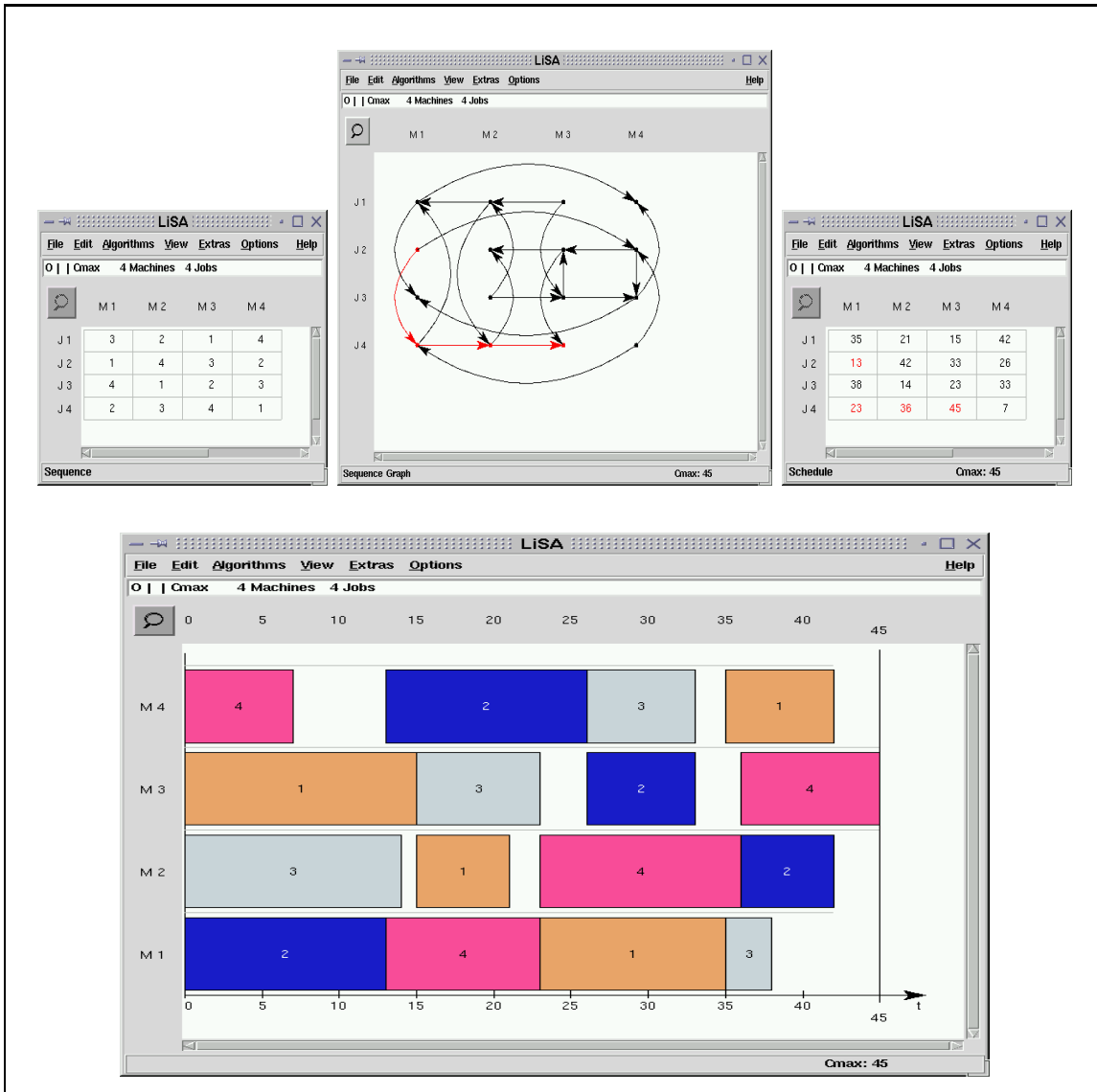


Figure 7: Output of LiSA

determines the complexity status (buttons **Extras**, **Problem Classification**). Moreover, the user can also get a complete reference by pressing the corresponding button. Another extra is the manipulation module. If you like to manipulate the schedule you can do so by pressing the mouse (right) on a certain operation of the Gantt chart. This operation can be moved one position earlier or later in both, the machine order or the job order, respectively. LiSA notifies, if the new combination of machine orders and job orders is not feasible. Each moving of an operation to a source or to a sink of the sequence graph generates again a feasible sequence and the corresponding Gantt chart is presented.

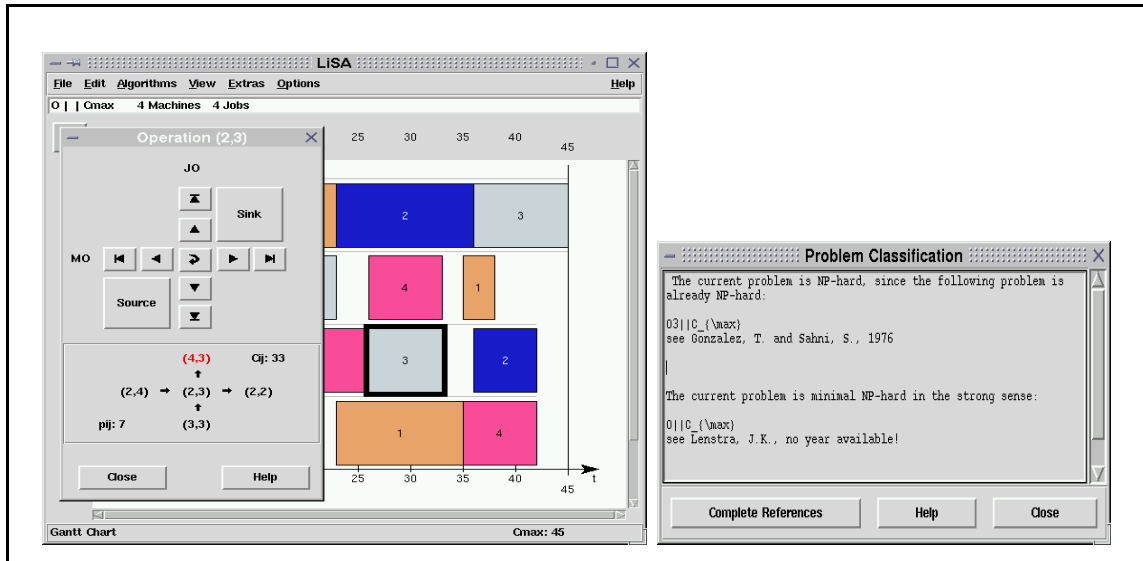


Figure 8: Extras of LiSA

7 File Format

Files are used by LiSA to save the type, an instance and solutions for a given scheduling problem. A file written by LiSA will have the extension LSA. Three special, very similar file types exist, two of those are used for communication with the external algorithms. The general format will be discussed at first here.

When saving a scheduling problem, LiSA writes as much data as there is available. That is first the type of the problem, then the current instance if one is defined and finally the current solution if one exists. Multiple solutions are not written by now.

Reading data from a file works just the same. The input file may contain either only a problem type, both the type and an instance, or the type, an instance and one or more solutions. The data in the file has to be in exactly the order it is written by LiSA, that is first the problem type, then the problem instance and finally the solution. We will now have a closer look at these components.

Generally LiSA uses whitespaces (space, tab, newline, etc.) as token delimiters. They are therefore crucial, especially between file entries and double opening or closing brackets.

Problem Type

The problem type is described in the usual $\alpha|\beta|\gamma$ notation. A file entry has to be in the following form:

```
<PROBLEMTYPE>
  Lisa_ProblemType= { [alpha] / [beta] / [gamma] }
</PROBLEMTYPE>
```

Figure 9: Problem type in a LiSA file

Note, that most of the following parameters are introduced in Section 4. For more information use the book of BRUCKER [5].

The **[alpha]** component describes the machine environment. It starts with exactly one of the following machine types: **1** , **0** , **F** , **J** , **X** , **G** , **P** , **Q** , **R** , **F;R1:** , **P;S1:** . This may directly be followed by **MPT** or **MPM** and finally a small **m** to indicate a fixed but arbitrary machine number or a **[number]** to give a fixed machine number.

The **[beta]** component describes job characteristics and additional constraints. One or more of the following parameters can be given, one from each group. Parameters may directly be followed by **a** ; or **,** but are separated by whitespace.

preemption: **pmtn**
precedence constraints: **intree** , **outtree** , **tree** , **sp_graph** , **chains** , **prec**
release dates: **r_i**
due dates: **d_i**
processing times: **p_ij=1** , **p_ij=p**
batch: **s-batch** , **p-batch**
bounded batch: **b<n**
job number: **n=k** , **n=[number]**
no wait: **no-wait**
size: **size_i**
time lags: **prec(1)** , **prec(l)** , **prec(l_ij)**
transportation delays: **t_ik=T** , **t_ikl=T** , **t_kl=t_lk** , **t_ikl=t_ilk** , **t_i** , **t_k** , **t_kl** , **t_ik** , **t_ikl**
server flags: **s_i** , **s_i=1** , **s_i=s**

The **[gamma]** component finally describes the objective function. It has to be exactly one of the following: **Cmax** , **Lmax** , **SumCi** , **SumWiCi** , **SumUi** , **SumWiUi** , **SumTi** , **SumWiTi** , **Irreg1** ,

LiSA supports a huge set of problem types (and the classification of them) but the main data structures and algorithms were developed for deterministic shop scheduling problems. You will therefore not always be able to create instances.

7.1 Problem Instance

An instance of a problem is given by the numbers of machines and jobs and various other data, like processing times, release dates etc. Please make sure that all the data needed by your problem type are enclosed. A file entry has the form given in Figure 10.

Here **m=** is the number of machines and **n=** is the number of jobs. It is crucial that these are the first entries, the rest may follow in arbitrary order. The matrices **PT=** and **SIJ=** of size **n × m** should always be in the entry. **PT=** contains the processing times for each operation while **SIJ=** indicates with entries **1** and **0** whether an operation exists or not.

The **M0=** matrix of size **n × m** contains the machine orders for job shop problems and flow shop problems.

The vectors **RD=**, **DD=**, **WI=** and **WI2=** of size **n** contain release dates, due dates, weights and additional weights for each job.

The **EXTRA=** vector of size **5** contains additional internal information.

```

<VALUES>
  m= [number]
  n= [number]
  PT= [matrix]
  MO= [matrix]
  SIJ= [matrix]
  RD= [vector]
  DD= [vector]
  WI= [vector]
  WI2= [vector]
  EXTRA= [vector]
</VALUES>

```

Figure 10: Instance in a LiSA file

A [matrix] of size $n \times m$ consists of n [vector]'s of size m in the following form:

```

{
  [vector]
  [vector]
  [...]
  [vector]
}

```

where a [vector] of size n consists of n [number]'s in the form:

```

{ [number] [number] [...] [number] }

```

Solution

In a schedule all important properties of a valid solution for a given problem instance are described. Therefore, it has to fit the problem instance defined in the file. You can define an arbitrary number of schedules in a file. A schedule file entry, see Figure 11, is very similar to the values entry.

```

<SCHEDULE>
  m= [number]
  n= [number]
  semiactive= [number]

  LR= [matrix]
  NMO= [matrix]
  NJO= [matrix]
  CIJ= [matrix]
</SCHEDULE>

```

Figure 11: Solution in a LiSA file

Again $m=$ is the number of machines and $n=$ is the number of jobs. **semiactive=** indicates whether the schedule is semiactive or not. It is crucial that these are the first entries, the rest may follow in arbitrary order. The matrices $LR=$ and $CIJ=$ of size $n \times m$ should always be in the entry. Recall, the sequence $LR=$ contains the rank for each operation in the sequence

graph and $CIJ=$ is the matrix of completion times.
 $NMO=$ and $NJO=$ are the corresponding machine order matrix and job order matrix.
 For the example considered in Section 6 the LSA file might look like in Figure 12.

```

<PROBLEMTYPE>
  Lisa_ProblemType= { 0 / / Cmax }
</PROBLEMTYPE>
<VALUES>
  m= 4
  n= 4
  PT= {
    { 12  6 15  7 }
    { 13  6  7 13 }
    {  3 14  8  7 }
    { 10 13  9  7 }
  }
  SIJ= {
    { 1 1 1 1 }
    { 1 1 1 1 }
    { 1 1 1 1 }
    { 1 1 1 1 }
  }
</VALUES>
<SCHEDULE>
  m= 4
  n= 4
  semiactive= 1
  LR= {
    { 2 1 3 4 }
    { 4 2 1 3 }
    { 1 3 4 2 }
    { 3 4 2 1 }
  }
  CIJ= {
    { 18  6 33 40 }
    { 41 13  7 27 }
    {  3 28 41 14 }
    { 28 41 16  7 }
  }
</SCHEDULE>

```

Figure 12: LiSA file for the considered example

Note, that anything between file entries is treated as comments. Of course, the starttags of fileentries $\langle PROBLEMTYPE \rangle$, $\langle CONTROLPARAMETERS \rangle$, $\langle VALUES \rangle$ and $\langle SCHEDULE \rangle$ are not allowed as comments.

Special File Types

There are two other, special file types used by LiSA to communicate with external algorithms. The algorithm input file is the same as described above, with an additional `<CONTROLPARAMETERS>` entry containing parameters necessary for the used algorithm. It is then located between the problem type and the values entry. The algorithm output file should only contain schedule entries, fitting the instance of the problem that was given to the algorithm.

8 Module interface

A module binary takes two arguments from the command line: an input file and an output file. Data will completely be exchanged over this two files specified in the command line. The input file has to be a LSA file containing the problem type, a set of control parameters and the values for a specific problem instance. It is important that these data appears exactly in this order. The output file is also a LSA file containing a schedule with the computed solution, see also Section 7.

For the module it is possible and desired too, that some information about the current state is given to the caller. This comprises information about the progress of calculation, the module process id and occurred errors or warnings. To give this information, a module should write "PID=", "OBJECTIVE=", "ERROR:" or "WARNING:" followed by the corresponding information to the standard output. This information will be parsed and handled by the main GUI, if the module was started from within it.

9 How to insert own algorithms

It is easy to extend LiSA by writing and binding in own modules. You have to write the module, specify the problems, it is responsible for, and supply a help file for each supported language, which are English and German at the moment.

9.1 Module binary

The module itself only must take care of its first two command line arguments, namely the input and output file names. After its execution there should be an output file in the right format. Additionally it can output some information about its state respectively progress.

It does not matter, which programming language is used for creating the module, as long as the module behaves correctly from an outer point of view. But if C++ is used, which is also the LiSA kernel programmed in, module creation is simplified heavily, since there already exist routines for LSA file access and LiSA data structure handling.

The completed module has to stay in `#{LISAHOME}/bin/`, so that the LiSA GUI can make use of it. Here is the source code for a simple sample.

```

// The name of the algorithm, the owner of this file
// and the date of the last changing:

/*
***** sample.cpp *****

Sample how to implement an algorithm for LiSA

Owner: LiSA

30.04.2001
/

// Include the header files for the used objects! Standard:
// LisaProblemtype requires global.hpp.
// The file global.hpp is needed by LisaProblemtype.
// The files ctrlpara.hpp, ptype.hpp and lvalues.hpp
// are used for parsing the input file.
// The object Lisa_Schedule (header: schedule.hpp)
// is necessary for writing the result into the output file.
// The file except.hpp is used for the exception handling.

#include <unistd.h>
#include <iostream.h>
#include <fstream.h>

#include "../main/global.hpp"
#include "../lisa/ctrlpara.hpp"
#include "../scheduling/schedule.hpp"
#include "../lisa/ptype.hpp"
#include "../lisa/lvalues.hpp"
#include "../misc/except.hpp"

int main(int argc, char *argv[])
{

// print a message that the program has started:
cout << "This is the LiSA Sample Module" << endl;

```

```

// The Lisa_Exceptionlist is forced for writing
// error messages to cout. Then LiSA is able
// to show the error messages of an external module.
G_ExceptionList.set_output_to_cout();

// open files and assure existence:
if (argc != 3)
{
    cout << "\nUsage: " << argv[0] << " [input file] [output file]\n";
    exit(1);
}

// tell LiSA the PID of this module's instance
cout << "PID= " << getpid() << endl;

// Define the objects for the file communication
Lisa_ProblemType * lpr = new Lisa_ProblemType;
Lisa_ControlParameters * sp = new Lisa_ControlParameters;
Lisa_Values * my_values=new Lisa_Values;

// The both parameter of the module call are the name of the
// input file and the name of the output file:
ifstream i_strm(argv[1]);
ofstream o_strm(argv[2]);

// read problem description, the controlparameters and
// the problem instance:
i_strm >> (*lpr);
i_strm >> (*sp);
i_strm >> (*my_values);

// Define the LiSA schedule object for storing results
Lisa_Schedule * out_schedule = new Lisa_Schedule(my_values->get_n(),
                                                my_values->get_m());
out_schedule->make_LR();

// *****
// ***** Insert your algorithm here: *****
// *****

int stages=MAX(my_values->get_n(),my_values->get_m());
int i,j;

```

```

for (i=0; i <my_values->get_n(); i++)
  for (j=0; j <my_values->get_m(); j++)
    if ((*my_values->SIJ)[i][j])
      (*out_schedule->LR)[i][j]= ((i+j) % stages)+1;

// The following lines demonstrate how to write into an extra LiSA window:
// cout << "WARNING: The Problemtyp is:" << lpr->output_problem()<< endl;
// cout << "WARNING: The upper bound is:" << sp->get_double("UPPER_BOUND")<<
// endl;
// cout << "WARNING: P(1,2)=" << (*my_values->PT)[1][2]<< endl;
// cout << "WARNING: Name:" << sp->get_string("NAME")<< endl;

// *****
// ***** End of Algorithm *****
// *****

// The object out_schedule contain the result of this algorithm,
// which is written into the output file
o_strm << *out_schedule;
delete out_schedule;
delete my_values;
delete lpr;
}

```

Figure 13: The file sample.cpp

Note, that in this sample an open shop problem with unit processing times and makespan minimization is to solve. The algorithm has only to construct a latin rectangle with maximal entry $\max\{n, m\}$ which is both: sequence and schedule. Clearly, this will be optimal in the case of $SIJ = I \times J$ and the sequence is a heuristic solution for arbitrary processing times.

9.2 The algorithm description file and the help file

For use with LiSA you have to classify your own algorithm. That is, you have to specify, which problem it can be applied to either as exact or heuristic technique. Moreover you should define, whether it is a constructive algorithm or an iterative one and which parameters it takes. Additionally, an HTML module description respectively help file can be declared. This information is held in an ALG file, which is LiSA's algorithm description file. There has to exist such a file for each supported language, because the algorithm's description must be mentioned in it.

The ALG files should reside in $\{\text{LISAHOME}\}/\text{data}/\text{alg_desc}/\text{language}/\{\text{LANGUAGE}\}/$ where $\{\text{LANGUAGE}\}$ is currently one of english or german. The English ALG file of the sample module is shown in Figure 14.

For online help about a module within LiSA this module must provide a help file. This help file should be written in HTML. There should be one for each supported language, which has to stay in $\{\text{LISAHOME}\}/\text{doc}/\text{lisa}/\{\text{LANGUAGE}\}/\text{algorithm}/$.

```
<GENERAL>
Name= sample algorithm
Type= constructive
Call= sample
Code= external
Help= algorithm/sample.html
</GENERAL>

<EXACT>
<PROBLEMTYPE>
Lisa_ProblemType= { 0 / p-ij=1 / Cmax }
</PROBLEMTYPE>
</EXACT>

<HEURISTIC>
<PROBLEMTYPE>
Lisa_ProblemType= { 0 / / Cmax }
</PROBLEMTYPE>
</HEURISTIC>

<PARAMETERS>
string TEST_PARAMETER ( TRUE FALSE ) "test parameter"
</PARAMETERS>
```

Figure 14: The file sample.alg

9.3 Module integration

If you program your module in C or C++ you cannot only make use of LiSA's data structures, but of LiSA's compilation and installation mechanism. In order to do this you have to create a compatible makefile containing the module name and a list of source files your module consists of. The makefile of the sample module and the source file list for the sample module, stored in the file `Make.List`, are given in in Figure 15 and in Figure 16 on the next page.

These files both should reside in $\{\text{LISAHOME}\}/\text{src}/\text{algorithm}/\{\text{PROGRAMNAME}\}/$ together with the sources and the ALG and help files for the different languages. $\{\text{PROGRAMNAME}\}$ denotes the module name in this context. The whole contents of the sample module source directory is shown in Figure 17.

The different language versions of the ALG and help files are denoted by special file names containing the language. You must stick to this file name patterns for having the installation process work properly.

```

# LiSA Sample Algorithm Makefile

# -----

# LiSA part:  sample

PROGRAMNAME=sample

# -----

TOPPROGRAMPATH=../../..

# -----

include ../Make.Algorithm

```

Figure 15: Makefile of the sample module

```

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

```

Figure 16: The file Make.List of the sample module

```

eldeh@legolas /LiSA/src/algorithm/sample > ll

drwxr-xr-x  3 eldeh  users  1296  2003-02-27  23:51  .
drwxr-xr-x 20 eldeh  users   864  2002-11-22  03:21  ..
drwxr-xr-x  2 eldeh  users   128  2002-11-22  03:23  CVS
-rw-r--r--  1 eldeh  users  5195  2002-09-06  22:15  Make.Depend
-rw-r--r--  1 eldeh  users   320  2002-11-22  03:23  Make.List
-rw-r--r--  1 eldeh  users   309  2002-09-06  22:15  Make.Objects
-rw-r--r--  1 eldeh  users   523  2002-11-22  03:23  Makefile
-rw-r--r--  1 eldeh  users  3527  2002-11-22  03:23  sample.cpp
-rw-r--r--  1 eldeh  users   381  2002-11-22  03:23  sample_english.alg
-rw-r--r--  1 eldeh  users   910  2002-11-22  03:23  sample_english.html
-rw-r--r--  1 eldeh  users   385  2002-11-22  03:23  sample_german.alg
-rw-r--r--  1 eldeh  users  1142  2002-11-22  03:23  sample_german.html

```

Figure 17: Directory of the sample module

After a "make depend" two more files (Make.Objects and Make.Depend) will be created automatically. Now the module can be compiled and installed together with all other LiSA modules.

10 Cooperative Development

One of the most important aspects to mention about LiSA is the way it is developed. Since the very beginning LiSA's developer team consists of a changing set of people, mainly students, distributing the code for the various parts modules of this software package. This dynamic as well as upcoming cooperations with external research groups providing legwork gave rise to the necessity for an environment being able to deal with this kind of cooperative development.

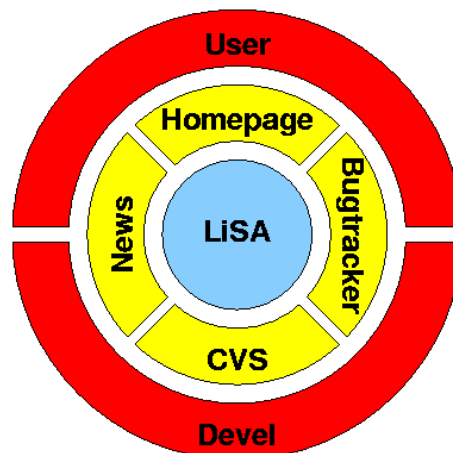


Figure 18: The cooperative development environment

Figure 18 shows the structure of this cooperative development: The main components and how both, user and developer are involved.

The central contact point for the user is the **homepage** of LiSA, see Figure 19. Here the user can download latest patches and new releases as well as documentations for the software and news of the research group. The homepage is also the main point for the user to get in touch with the developers: Pages with brief descriptions of the current and former members of the LiSA team give details about responsibilities and availability of each of them. In addition to that there is a direct link to the news server on the one hand and the bug- and feature-tracking system on the other, which in turn provide feedback from and back to the user.



Figure 19: LiSA homepage: <http://lisa.math.uni-magdeburg.de>

The **bug- and feature-tracking system** represents an essential component for the development of LiSA for both, user and developer. For the user it provides an interface to report bugs or ask for new features. He can track such a report and ask the system to get informed on any change of any or a particular type (e.g.: “acknowledged”, “resolved”) that has been applied to the report. The user is at any time able to check for the current state of his report, the developer it has been assigned to and the remaining estimated time for resolution. Finally he can always add further notes to support the developer in his work. Altogether the bug- and feature-tracking system makes the progress of development highly transparent for the user.

For the developer in turn it basically embodies an enhanced kind of todo-list. Every incoming bug report or feature request is assessed, acknowledged and finally assigned to the most appropriate developer. The assessment primarily covers a check whether this bug has

already been reported before and the reproducibility followed by a rating of severity and priority. The developer it is assigned to will additionally assess projection (e.g. “redesign” or “major rework”) and give an estimation of the time necessary for resolving the bug or implementing the feature.

The bug- and feature-tracking system provides an easy-to-use overview of all outstanding tasks. On the one hand every developer knows at any time exactly which bugs and feature requests are currently assigned to him. On the other hand the priority rating of the reports allows a scheduling of the remaining work - the number of high-prioritized jobs for example might indicate the amount of work that has to be done before the next version can be released.

The **version control - & source code management system** is an internal mechanism for the developers to coordinate their work. The high number of people distributing code for the software package makes it impossible to collect all the various and frequent changes and extensions and merge them by hand. Thus it is inevitable to take use of a version control system that provides source code management based on a centralized data storage container. Every developer still has his own working copy, but frequently updates his “sandbox” bi-directionally against this unique master copy, called “repository”. The main advantage of this proceeding is that changes are quite unlikely to get lost: Once a developer committed a change to the repository it is stored there permanently - and the other way round, keeping on requesting the latest changes from the repository will ensure the local working copy always to be up-to-date.

The possibility to track all changes ever done on the code base enables the developers to easily isolate code that introduced new bugs from one version to another.

A last advantage to mention is the freedom provided by the CVS to develop in different branches in parallel without interacting with influencing the code outside the particular trunk. This way it is possible, for instance, to work out bugfixes for the current release while already developing on the next version. Those fixes can easily be merged into the new version at any time later on...

Finally the **news** system provides a gateway for feedback back to the user. Here upcoming releases and patches will be announced, frequently asked questions will be answered and latest results of the research group will be posted. On the other hand, the news system is also intensively used for discussing matters of internal concern. This covers coordination of main stream development as well as intra-cooperative communication...

References

- [1] Baker, K.R. [1984]: *Introduction to Sequencing and Scheduling*; Wiley & Sons, New York
- [2] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J. [1996]: *Scheduling Computer and Manufacturing Processes*; Springer Verlag Berlin-Heidelberg-New York
- [3] Bräsel, H.[1990]: *Latin Rectangle in Scheduling Theory*; Professorial Dissertation (in German), University Magdeburg, Germany
- [4] Bräsel, H./Tautenhahn,T./Werner,F.[1993]: *Constructive Heuristic Algorithms for the Open Shop Problem*; Computing, 51, 95-110

- [5] Brucker, P. [2001]: *Scheduling Algorithms*; Third Edition, Springer Verlag Berlin-Heidelberg-New York
- [6] Chretienne, P., Coffman, E.G., Lenstra, J.K. (Editors) [1995]: *Scheduling Theory and its Applications*; John Wiley & Sons, Chichester-New York-Brisbane
- [7] Conway, R.W., Maxwell, W.L., Miller, L.W. [1967]: *Theory of Scheduling*; Addison-Wesley Publishing Company, Massachusetts
- [8] Dannenbring, D.G. [1977] *An Evaluation of Flowshop Sequencing Heuristics* Management Science 23, 1174-1182
- [9] Domschke, W., Scholl, A., Vo, S. [1993]: *Produktionsplanung - Ablauforganisatorische Aspekte*; Springer Verlag Berlin-Heidelberg-New York
- [10] French, S. [1982]: *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*; John Wiley & Sons, New York
- [11] Adams, J., Balas, E., Zawack, D. [1988] *The Shifting Bottleneck Procedure for Job Shop Scheduling*; Management Science 34, 391-401
- [12] 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
- [13] 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-New York-Tokyo
- [14] Lageweg, B.J., Lawler, 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
- [15] Lenstra, J.K. [1977]: *Sequencing by enumerative methods*; Centre Tracts 69, Amsterdam
- [16] Morton, T.E., Pentico, D.W. [1993]: *Heuristic Scheduling Systems*; John Wiley & Sons, Inc., New York
- [17] Muth, J.F., Thompson, G.L. (Editors) [1963]: *Industrial Scheduling*; Prentice Hall, Englewood Cliffs
- [18] Pinedo, M.: *Scheduling [1995]: Theory, Algorithms and Systems*; Prentice Hall, Inc., Englewood Cliffs, New Jersey
- [19] Rinnooy Kan, A.H.G. [1976]: *Machine scheduling problems: Classification, Complexity and Computations*; Martinus Nijhoff/ The Hague, 1976
- [20] Tanaev, V.S., Gordon, V.S., Shafransky, Y.M. [1994]: *Scheduling Theory: Single-Stage Systems*; Kluwer Academic Publishers, Dordrecht
- [21] Tanaev, V.S., Sotskov, Y.N., Strusevich, V.A. [1994]: *Scheduling Theory: Multi-Stage Systems*; Kluwer Academic Publishers, Dordrecht

Appendix: License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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 referring 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. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee 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 transferring 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 licensees 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, mere 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 preferred 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 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 licensee 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