Production schedules using timed automata

Report for the joint advanced student school 2008 - Advanced Methods for intelligent automation and control

Dipl.-Ing.(FH) Inga Krause

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Olaf Stursberg

Coursedirectors: Prof. Dr.-Ing. Olaf Stursberg and Prof. Dr. V.P. Shkodyrev
Tutors: Dipl.-Ing. Tina Paschedag and M. Sc. Hao Ding

Contents

List of Figures

Chapter 1

Introduction to Scheduling

For processing or manufacturing systems intelligent scheduling algorithms help to optimize the desired output. A system can be devided into resources and production steps (operations). The operations require a resource for a specific time. Since there is a finite number of resources available in a plant, the resources represent a limiting constraint. It is the task of scheduling to determine in which order operations are allowed to allocate resources such that the profit is maximized or the cost minimized. Cost reduction can be achieved for example by either minimizing the makespan or maximizing the throughoutput of the plant. These cases are useful for cyclic operation schemes in which the plant should produce as much as possible in a short period of time. From a hierarchical point of view, the scheduler produces high level commands to the process which triggers a subordinated sequential controller. This subordinated controller in turn realizes the resource allocation, the execution of the operation and eventually the de-allocation of the resource. A variety of systems like telecommunication systems and real-time operation systems require scheduling and have been already intensively investigated in the last decades. The main idea in the scheduling algorithm was to calculate lower cost bounds as a mean to exclude non-optimal schedules. One approach was solving subproblems with relaxed integrality constraints by linear programming, for example the mixed-integer linear programming (MILP) (presented by J. Kalrath [Kal02] and E. Kondili, C. Pantelides and R. Sargent [EKS93]). Alternatives to the linear programming are given by constraint programming, genetic algorithms or reachability algorithms for models given as so called timed automata (TA). TA will be explained in detail later. They provide an intuitive modular and graphical form by assigning resources to nodes and having transitions between them. Additional techniques extend the reachability analysis for TA by the notion of costs and optimal executions and make the algorithm more efficient also for larger problem instances. S. Panek, O. Stursberg and S. Engell embeded linear programming in TA in order to obtain lower cost bounds for partial executions of the TA ([SPE04b] and [SPE04a]). This theory has been extended by the same authors by further reduction of the solution space on the fly ([SPS] and [SPE06]). The further reduction is achieved through non-

lazy executions, the sleep set method and embedded linear programs are updated iteratively. This latter approach with its components and experimental results will be mainly explained in this report. But first of the basic concepts of scheduling which are job-shop scheduling, timed automata with its reachability analysis and schedule-modelling will be explained.

1.1 Job-Shop Scheduling

The alogrithm which will be presented is designed to solve job-shop scheduling problems. A typical job-shop problem is defined as follows: a set of operations O is assigned to a set of Resources R with the function v: $O \rightarrow R$. A mapping d assigns the duration and the schedule s a start time to each operation O. The mapping κ assigns operations to jobs with the number of jobs being smaller or equal tne number of operations. For the following considerations a schedule is considered to be valid if it satisfies the following requirements:

1. Once an operation has started it has to be executed continuously and cannot be preempted

2. Once an operation is executed on a resource no other operation can be assigend to this resource during the operation time

3. κ determines the order of operations within a job and this order cannot be changed during execution

The scheduler has to determine a schedule which is optimal with respect to a given cost or profit function. Costs and profits can be assigned to various elements, for example the operations, the resource utilization, setup and changeover procedures, material stocks in storages, production delays with respect to deadlines, elapsed time, etc. For the following algorithm only the minimizing of the makespan, which is the finishing time of the job is considered. Note that if the waiting times between operations are not restricted there would be an infinite set of valid schedules.
General scheduling problems are often like as well in this case limited to job-shop scheduling because only for this case efficient algorithms for medium- to large-size problems do exist.

1.2 Timed Automata

Timed automata (TA) were developed by Y. Abdeddaim and O. Maler [AM]. As already mentionned, in a graphical form they assign resources to nodes, having transitions between them. They are a useful tool for scheduling.

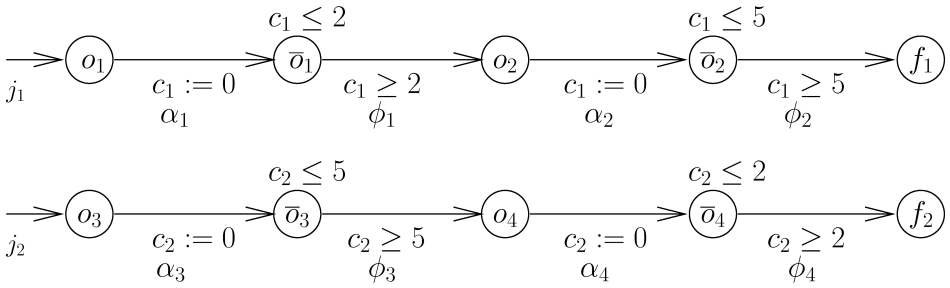

Figure 1.1: Model of two parallel Job-Timed-Automata

Figure 1.2: Model of twoe parallel Resource-Timed-Automata

1.2.1 Job-Shop Scheduling Modeled by TA

A job-shop scheduling problem can be described by TA in the case of makespan minimization. There will always be a job automata to solve the scheduling problem. This job automata consists a chain of nodes representing the different operations. There are two kind of nodes/location for each operation: one at which the job waits until the required resource becomes available (l_{ij} - for the job i and the operation step j) and another one $\bar{l}_{ij}$ at which the operation occupies the resource for the duration d(o_{ij}). An additional final location f_i indicates that the job is finished. Clocks c_i monitor the duration for which an operation occupies a resource. There are two kind of transitions: one to start an operation with the label α_{ij} and one to finish it with the label ϕ_{ij}. Figure 1.1 shows the model of two parallel working TAs. One method to enable mutual exclusion in resource utilization is to develop a separate resource automata. Each resource needs a resource automata with the two locations idle (i) and busy (b). The same labels α_{ij} and ϕ_{ij} are connected to the transitions as for the job automata which achieves synchronisation (see in figure 1.2).
To this solution for mutual exclusion additional timing or capacity constraints can be added.

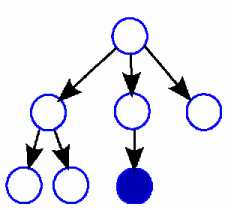

Figure 1.3: Tree Structure for Reachability Analysis

1.2.2 Formulation of TA and its Semantics

According to S. Panek, S. Engell, and O. Stursberg [SPE06], a TA is defined as a tuple (L, l_0, E, inv) with a finite set L of locations including the initial location l_0, an invariant inv: $L \rightarrow B(\mathbb{C})$ and the set $E \subset LxB(\mathbb{C})xActxP(\mathbb{C})xL$ of transitions. Act denotes a set of actions, $B(\mathbb{C})$ are constraints formulated for a set $\mathbb{C}$ of clocks, and $P(\mathbb{C})$ is a set of reset assignments. The constraints in $B(C)$ are given as conjunctions of atomic formulae $x \sim n$ or $x - y \sim n$ with $x, y \in \mathbb{C}, \sim \in \{<, \leqslant, =, \geqslant, >, \}, n \in \mathbb{N}$. A transition $E \in (l, g, a, r, l')$ with a guard $g \in B(\mathbb{C})$, $a \in Act$ and $r \in P(\mathbb{C})$ between a source location l and a target location l' is denoted by $l \xrightarrow{g,a,r} l''$.
Figure 1.1 shows a set of locations L (o_i) with its invariants (for example $c_1 \leqslant 2$) and the transitions with the reset assignments P ($c_i := 0$) and the set of labels Act (α_i and ϕ_i).
The authors further described the semantics of an TA A as a labeled transition system $(\mathbb{Q}, (l_0, u_0), \Delta)$ with the state space $\mathbb{Q}$ consisting of pairs (l, u) with u being a vector of valuations of the clocks in $\mathbb{C}$. The pair (l_0, u_0) represents the inital state and Δ denotes a transition relation with the following rules:

1. Progress of time: $(l, u) \rightarrow (l, u + d)$ if $(u + e)$ satisfies $inv(l) \forall 0 \leqslant e \leqslant d$,
2. Transition: $(l, u) \xrightarrow{a} (l', u')$ if $l \xrightarrow{g,a,r} l'$ exists in E for A such that $g(u)$ evaluates to true and $u' = r(u + d)$.

1.2.3 Reachability Analysis and Scheduling for TA

Efficient algorithms exist to determine logic properties of TA by exploring the reachability of TA (see figure reffig:Tree). The tree encodes reachable states symbolically as pairs (l,Z) of locations l and zones Z. Z is an infinite set of possible clock valuations in l and is expressed as a conjunction of finitely many inequalities. The reachability task has been extended in the past years by considering costs as well. The objective ist to find a path such that the cost is minimized. In the case of makespan minimization, the right trace is found when getting the infimum of all possible traces. There are several tools which can determin the optimal trace of TA: Uppaal CORA (based on so-called priced zones), IF and TAopt.

Chapter 2

Concepts of the Algorithm

The following algorithm is suitable for minimizing the makespan of job-shop scheduling problems modeleld by TA. It includes the reachability algorithms for timed automata supplemented by a cost evaluation of traces (as mentionned in 1.2.3).

2.1 The Scheduling Algorithm

This scheduling algorithm embeds the generation of lower bounds for the cost-to-go by using linear programming which can be used to prune the search tree efficiently. In addition priorities are calculated for the nodes to search towards the optimum and the search tree is reduced on-the-fly as can be seen in the algorithm shown in figure 2.1. The directed graph is therefore shortened after the so-called branch-and-bound principle.
The input to the algorithm is a TA model A, an LP model M, an initial location l_0 and a target location l^*. In a target location all operations are scheduled. The LP model M is an algebraic mixed-integer formulation with relaxed integrality constraints, as will be described in 3.1. The algorithm operates on a waiting list W, a passed list P, and a successor list S. The elements of these lists are nodes defined as six-tuples (l, u, b, c, d, p) refering to a symbolic state as follows: l denotes the location, u the vector of clock valuations upon reaching the location, b an underestimate of the costs encountered for reaching the final state (all jobs done), c the accumulated cost of reaching the current state, d the depth of the current node (i.e. the number of predecessors along the path starting from l_0), and p an auxiliary priority for selecting elements from the waiting list. The cost c represents the minimal time to reach a destination l in the case of makespan minimization with c^* being the minimum found cost which is an upper bound for the graph. According to the branch-and-bound principle, the branches of the search tree are cut that does not lie within the lower bound b and upper bound c^*. Figure 2.2 illustrates this principle: a path along the successors (in yellow) of the current state at depth $d = 3$ (in blue) is cut if $c + b > c^*$. The algorithm contains the following functions:

$c^* := \infty$; $c_0 := 0$; $\mathcal{P} := \emptyset$; $\mathcal{W} := \emptyset$; $\mathcal{S} := \emptyset$
$(b_0, x) :=$ SolveLP(M)
$p_0 :=$ ComputePriority$^*(b_0, c_0, 0, x)$
$\mathcal{W} := \{(l_0, \mathbf{0}, b_0, c_0, 0, p_0)\}$
WHILE $\mathcal{W} \neq \emptyset$
    $(l, u, b, c, d, p) :=$ SelectRem$(\mathcal{W})$
    IF $l = l^*$ THEN
        IF $c < c^*$ THEN
            $c^* := c$
        END
    ELSEIF $b \leq c^*$ AND $c < c^*$ THEN
        $\mathcal{S} :=$ CompSucc$(A, (l, u, b, c, d, p))$
        $\mathcal{S}' :=$ FilterSucc$(\mathcal{S})$
        $\mathcal{S}'' :=$ InclusionTest$(\mathcal{P}, \mathcal{S}')$
        FOR ALL $(l', u', -, c', d', -) \in \mathcal{S}''$ DO
            $h :=$ ComputeHistory$(l', u', -, c', d', -)$
            $M^u :=$ UpdateM(M, h)
            $(b', x') :=$ SolveLP(M^u)
            $p' :=$ CompPriority(b', c', d', x)
            $\mathcal{W} := \mathcal{W} \cup \{(l', u', b', c', d', p')\}$
    END; END;
    $\mathcal{P} := \mathcal{P} \cup \{(l, u, b, c, d, p)\}$
END

Figure 2.1: The search algorithm

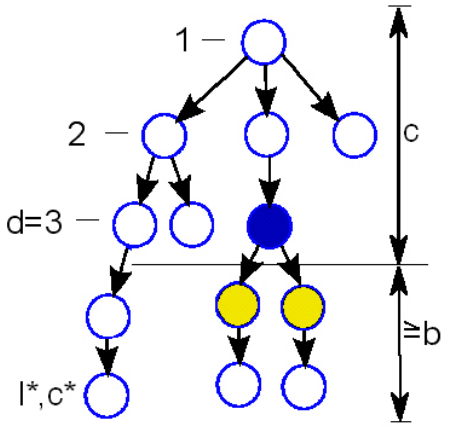

Figure 2.2: Branch-and-Bound Principle

- **SolveLP** solves the relaxed MILP model of the scheduling problem for the current model M by linear programming (LP), and returns the objective value b and the solution vector x which contains the values of the relaxed decision variables.
- **CompPriority** computes and assigns auxiliary priorities p to the nodes based on the quantities b, c, d and x.
- **SelectRem** selects and removes the six-tuple from the waiting list which has the highest priority using the various attributes of the nodes.
- **CompSucc** determines all successor states that are reachable from (l, u) using the selected six-tuple (l, u, b, c, d, p).
- **FilterSucc** removes those successor nodes from S which exhibit so-called laziness (this term will be explained in section 2.4.
- **InclusionTest** avoids to visit nodes again which have been already explored.
- **UpdateM** updates the LP model such that some or all (originally relaxed) variables in Mu are fixed to values that correspond to the history h. The latter is a partial trace from the initial to the current node. If l = l^*, all variables that encode the sequence are fixed to zero or one.

When the calculations in the algorithm come to an end because the target location is reached, P contains the optimal path.

2.2 Computation of Lower Bounds

The lower bounds indicate which path can be excluded because they are suboptimal. For the presented algorithm a tailor-made MILP model for the specific case of makespan minimization of job-shop scheduling is generated. The MILP model is tailored in the way that only binary variables encode true degrees of freedom but no additional variables to retain the automaton structure are used. This enables much faster LP solving and it is not reformulate the TA model in an algebraic model because it can be dereived directly from the TA. Figure 2.3. shows the overal structure of a tool which realizes the algorithm presented before in figure 2.1.

2.3 Search Heuristics

An important factor influenceing the performance of the algorithm is the search heuristic. Depending on the search heuristic the speed towards the optimum changes. The selection criterion for the following node is implemented in the mentioned function SelectRem. Selection criteria are based on the node attributes (b, c, d, p)

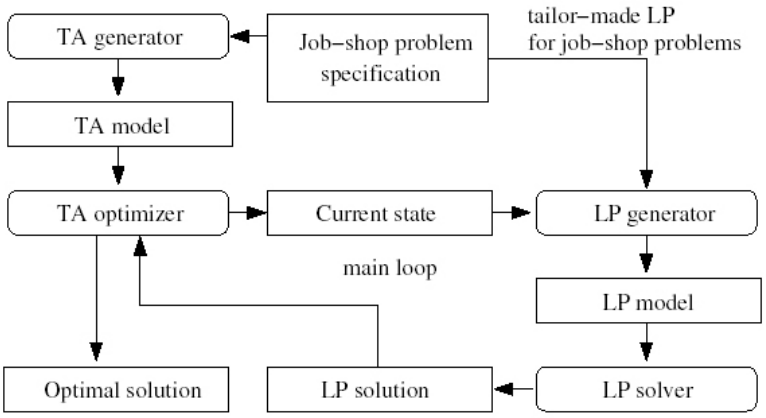

Figure 2.3: Structure of the Implementation

taking the maximum or minimum value of a chosen attributte. Figure 2.4 displays the different selection strategies.

Depth-First Search or Breadth-First Search	Min-Cost Search (Best-First Search)	Best-Lower-Bound Search	Random Search	Compute Priority
Select node with maximal/minimal depth d	Select node with minimal cost c	Select node with lowest bound b	Select on random distributed priorities p	Select node with highest priority p → evaluation of LP-solution vector x

Figure 2.4: Selection Strategies

It is often useful to combine different selection criterion so that if the first selection criterion (e.g. the depth-first strategy) selects more than one node, a second criterion evaluates this choise and so on until one single node is obtained. Experimental results of different simple and combined selection criteria will be presented in the section 4.3.

2.4 Non-laziness and Immediate Traces

The notion **non-lazy** and **immediate** traces are used to describe a certain behaviour of TA. These traces do not exhibit periods of useless waiting, arising for

example if none of a set of tasks is started whereas the required resources are available. For non-lazy traces exist a stricter criterion than for immediate traces as can be seen in figure 2.5. An optimal trace is always at least immediate in the case of makespan and tardiness minimization. For more general cost functions however, waiting before/after operations can be advantageous for example if a machine should not exceed a certain working temperature.

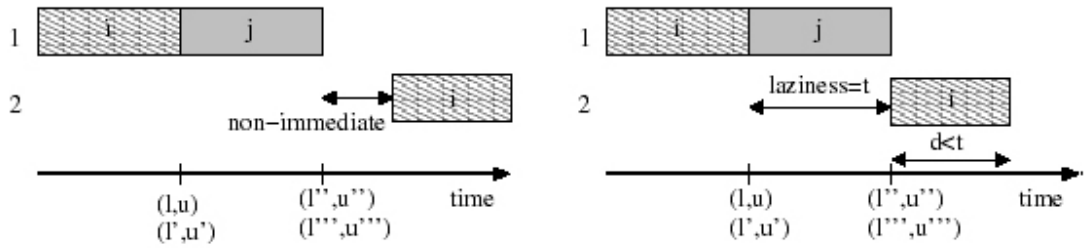

Figure 2.5: The left schedule is non-immediate, the right one is immediate but lazy

To achieve immediate traces unnecessary waiting time in the transition sequence $(l, u) \rightarrow (l, u + t) \rightarrow (l', u')$ with $t' < t$ need to be reduced and the following sequence is formed: $(l, u) \rightarrow (l, u + t') \rightarrow (l', u')$. Hence any time transition (which represents waiting) followed immediately by an allocating ransition can be omitted. The effect on the search of the symbolic state space is that the number of remaining time transitions become finite.
With the concept of non-laziness waiting in (l', u') in figure 2.5 is only allowed as long as the operation of job i on machine 2 with length d could not be started and finished in between. In other words, the waiting period t allowed in (l', u') is strictly limited by the length d of the shortest enabled operation by $t < d$. "Holes" which are large enough to be filled with an enabled operation are forbidden in the schedules. This strategy in which waiting is allowed under certain conditions is therefore also called **weakly non-lazy** strategy.
A even more restrictive condition with respect to waiting is **strong laziness**: whenever the list of successor states is not empty, waiting is forbidden in the current state (l, u). In this strategy as many operations as possible and as early as possible are scheduled on available resources. Only if no new operations can be started, waiting is permitted until the next running operation finishes.This method is also called greedy strategy.

Chapter 3

Algebraic Mixed-Integer Linear Programming Formulation

As already mentionned before, lower cost bounds are helpful to efficiently prune the reachability tree. For this purpose, the MILP model of the underlying scheduling problem is relaxed, modified to reflect the current state of the scheduled system, and solved. Relaxed means that an operation can be allocated for example 90% of the execution time of an operation is allocated on resource 1 and 10% of its execution time on resource 2. This is useful to calculate quickly a lower bound. However for the scheduling an operation is fixed to only one resource.

3.1 Algebraic MILP Formulation

The key principle is to use binary variables only to encode the possible sequences in which operations are processed on the same resource. This means that for each pair (o_i, o_j) of operations with $v(o_i) = v(o_j)$ one binary variable is defined, and its value describes which operation is processed first. In addition, continuous variables are used to encode starting dates of operations.
For the description of a job-shop scheduling problem the following variables, parameters and constants are needed:

- duration of the operations: d(o)
- overall time horizin H as a safe upper bound for the makespan
- positive continuous variables to store the starting dates of the operations: s(o)
- Binary variables to indicate the order in which the operations are executed on each resource with $p(o, o') = 1$ if o is executed before o' and $p(o, o') = 0$ otherwise.

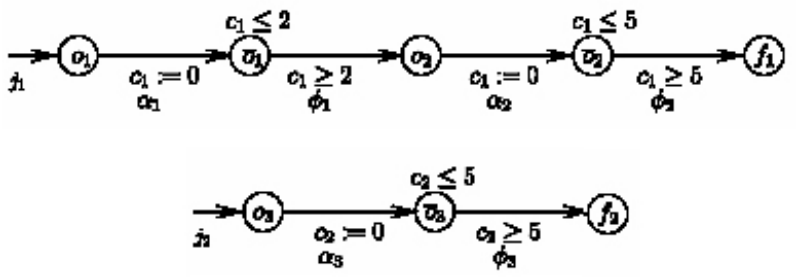

Figure 3.1: Timed Automata for MILP Formulation

In addition equations are needed to complete the algebraic form:

- Each operation must be started at or after time zero, and must be finished before $H : \forall o \in O : s(o) \geqslant 0, s(o) + d(o) \leqslant H$.
- All operations belonging to one job must be executed in the specified order: $\forall o_i, o_q \in O$ with $(o_i) = \kappa(o_q) \wedge i < q : s(o_i) + d(o_i) \leqslant s(o_q)$.
- Simultaneous execution of operations on one resource is excluded by: $p(o, o') + p(o', o) = 1$ for all operations assigned to one resource.
- The start and end times of the operations executed on a resource are ordere by the following condition: $s(o) + d(o) - s(o') \leqslant H(1 - p(o, o'))$ for all operations assigned to one resource and $o \neq o'$.
- The minimization of the makespan is written in (in-)equalities as $min\ \Psi$ such that $\Psi \geqslant s(o) + d(o)$ for all operations.

3.2 Example of the Algebraic MILP Formulation

For the small Timed Automata example shown in figure 3.1 in which o_1 and o_3 need the same resource the following equations are needed to develop an Algebraic MILP Formulation:

- Time horizon: $H = 100$
- Execution order: $s(o_1) + 2 = s(o_2)$
- Mutual exclusion: $p(o_1, o_3) + p(o_3, o_1) = 1$
- Makespan: $min\ \Psi$ with $s(\bar{o}_2) + d(\bar{o}_2) \geqslant \Psi$, $s(\bar{o}_3) + d(\bar{o}_2) \geqslant \Psi'$

Chapter 4

Experimental Results

To test the presented algorithm S. Panek, O. Stursberg and S. Engell employed two series of job-shop instances to investigate its performance. In this chapter a brief summary of the results will be given (more data in **??**).

4.1 Test Series

The first series (A) was produced by a random generation procedure which creates as many operations in the way that for each job there is one operation per resource. The duration of a resource is distributed randomly over 1,2..,6. The operations are randomly assigned to the resources for each job. The number of jobs are varied from 2 to 6 and the number of resources from 2 to 4. This instances are sufficiently small to explore the state space until the waiting list gets empty, and thus can guarantee to compute the optimal solution.

The second series (B) is a set of job-shop benchmark instances published in the operations research literature in the past 20 years.
In all tests the software tool called TAopt was used. It is an implementation of:

- cost-optimal reachability algorithm for TA employing branch-and-bound
- computation of lower bounds from embedded LP problems
- various node selection criteria (like Depth-First Search etc.)
- weak and strong non-laziness and the sleep set method

In order to solve the embedded LP problems, the commercial package Cplex 9.0 was used, and both test series were carried out on a 2.4 GHz Pentium 4 machine with 1 GB of memory, SuSE Linux 8.1.

Jobs/ Res	pure	bb	wnl	snl
2/2	31	12	27	25
3/2	214	87	218	95
4/2	1074	189	641	233
5/2	6343	2367	3155	1042
6/2	37553	15949	46943	5803
2/3	62	19	38	21
3/3	741	142	284	73
4/3	7445	1631	1165	366
5/3	87365	13843	34772	3392
6/3	549324	65934	316120	1182
2/4	126	26	72	39
3/4	2631	582	722	103*
4/4	35172	4677	5189	641
5/4	451290	28394	50163	3004
6/4	5253234	1032669	738293	9503*

Figure 4.1: Number of Explored Nodes using Different Techniques for State Space Reduction

4.2 Techniques for State Space Reduction

The pure standard depth-first method was compared to the standard depth-first method with branch-and-bound (bb) or with weak non-laziness (wnl) and strong non-laziness (snl) settings in terms of explored nodes (see figure 4.1). Results on series A show that the number of explored nodes significantly decreases when using branch-and-bound techniques with embedded linear programming. Another observation is that configurations involving non-laziness show that weak and strong non-laziness drastically reduce the number of explored nodes in most cases. However weak non-lazyness cannot use clock reduction techniques because of the need for clock evaluations for the employed condition $d \leqslant t$. This is the reason why not always fewer nodes were explored. Results reveal as well that combinations of individual methods (weak or strong non-lazyness and branch-and-bound) lead to an even stronger reduction of explored nodes.

(a)	min c	(b)	max d	(c)	max d, min c
(d)	max d, min b	(e)	max d, min b, min c	(f)	min b
(g)	min b, min c	(h)	min b, max d	(i)	min b, max d, min c

Table 4.1: Search Strategies

Jobs/Res	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)
2/2	21	12	12	12	12	12	12	12	12
3/2	96	42	42	42	42	58	83	42	42
4/2	211	89	89	**53**	**53**	61	98	**40**	**40**
5/2	843	415	415	**386**	**386**	422	519	**378**	**378**
6/2	4810	2562	2562	**1058**	**1058**	1202	1983	**1061**	**1047**
2/3	19	19	19	19	19	19	19	19	19
3/3	54	58	58	58	58	47	47	47	47
4/3	282	143	143	**114**	**114**	147	158	**114**	**114**
5/3	2417	831	831	**648**	**648**	1038	1177	**508**	**508**
6/3	12640	**1906**	**1906**	**1851**	**1851**	2677	3511	2118	2070
2/4	27	27	27	27	27	27	27	27	27
3/4	75	75	75	42	42	42	42	42	42
4/4	410	232	232	**99**	**99**	151	168	**99**	**99**
5/4	1628	638	638	**446**	**446**	493	721	**434**	**434**
6/4	6379	4045	4045	**3919**	**3919**	4079	6051	**3817**	**3794**

Figure 4.2: Number of Explored Nodes using Different Search Strategies

4.3 Search Strategies

The explored nodes using the search strategy combinations listed in table 4.1 are shown in figure 4.2.
It was observed that min c (best-first) criterion is not the method of choice because the number of explored nodes is quite high. Combined criteria are found to be preferable for the majority of tests and in particular the combination of min b and max d produced good results throughout.

4.4 Test for Benchmark Examples from Literature

For series B different tools were compared: Cplex for a pure MILP solving, TAopt for the presented approach of combining MILP solving to find a lower bound with TAs and Kronus a TA-based programm which does not consider lower cost bounds. CPU usage ws limited to 1 hour. TAopt was used as for series A with the combination of branch-and-bound and strong non-laziness. However instead of the following selection criterion for the waiting list: dipth-first-based strategy max d, min b, minc the best-lower-bound strategy min b, max d, min c was chosen because the first strategy didn't compute the optimal solution within 1 hour. TAopt was run with and without generation of lower bounds. The configuration without lower bounds led to a memory overflow such that the nodes had to be restricted to 10^7. With only one exception the configuration with lower-bounds produced a better solution (closer to the optimum) than the one without lower-bounds. The programm Cplex perforemd the best for small instances (e.g. 10 jobs and 10 resources) however for bigger instances (e.g. 15 jobs and 15 resources) the limited computation time prevents Cplex from finding better solutions, such that TAopt performed better. Also it was discovered that although TAopt needed slightly more time to find a first feasible solution the cost bounds are significantly lower for the remaining computation time. In comparison with Kronus half of the time TAopt finds slightly better solutions than those reported by Kronus. Some problem instances could not be solved by Kronus within the computation time of 1h.

Chapter 5

Extensions to More General Scheduling Problems

In industry however the simple structure of job-shop scheduling is not always sufficient to describe the situation. Sometimes alternative or parallel paths are needed or the the individual operations only need to follow a partial order. There might be timing constraints between operations which must be considered (e.g. time intervals in which an operation needs to be finished) or for changeover procedures for a material to another resource, the allocated resource doesn't execute an operation and could be configured in a different way during this time. The availabitlity of a resource might be lmited for example to day time only. Another type of usual constraint accounts for the fact that operations consume or produce materials. Hence a limitation on storage, resource capacity and batch size needs to be considered.
However TA are not restricted to deal only with job-shop problems. They can be altered so that most of the cases just mentioned can be dealt with:

- Alternative production paths can be described through another inserted possible operation-resource-pair.
- Parallel production paths are modeled by a parallel separate TA. Using additional shared binary variables mutual exclusion of operations of the same job is guaranteed.
- Timing constraints can be included by invariants in waiting locations and guards on outgoing transitions.
- For changeover procedures additional waiting locations with appropriate guards on their transition are added to ensure that the resource automation waits until the changeover period expires.
- Restricted working times can be handled by employing separate night-shift and weekend automata which are synchronized with the original automata amd blocking them at the times the machines should not run.

- The size of material stocks can be expressed by shared variables. The consumption of material and production of another material can be formulated by introducing guards and actions on the corresponding transitons.

Chapter 6

Conclusion

A quite recent approach to job-shop scheduling from S. Panek, O. Stursberg and S. Engell has been presented that combines reachability computations for timed automata with a branch-and-bound principle using lower cost bounds obtained from embedded linear programming, and with the sleep set method and the notion of non-lazy traces to exclude non-optimal solutions. The experimental results revealed that for larger problem instances solutions are found within a limited computation time that are comparable to or even better than those obtained with established MILP techniques. Hence the mixed TA/LP approach is prefered for good schedules of large problems within a short period of time. Another advantage of timed automata is the intuitive modular form in separate job and resource automata with a simple structure whereas finding a problem formulation that can be efficiently solved by mixed-interger programming tolls can be cumbersome and time-consuming task. In comparison to other TA-based methods (as implemented Kronos), the embedded linear programming step can be assessed as follows: it leads to a trade-off between an increased computation time per node, and a reduced number of nodes explored. The results obtained for the larger problem instances show that the benefits of an efficient pruning and of using the cost bounds to minimize the search are larger than the increase of computation time per node caused by the LP step. The effect of pruning also leads to smaller memory consumption, which is not a negligabe aspect because scheduling of large problems with tools for model checking often fails du to memory overflows and not to prohibitive computation time.

Bibliography

[AM] Y. Abdeddaim and O. Maler. Job-shop scheduling using timed automata. In G. Berry and H. Comon, editors, *Computer Aided Verification (CAV)*.

[EKS93] C. Pantelides E. Kondili and R. Sargent. A general algorithm for short-term sched. of batch operations - milp formulation. *Comp. Chem. Eng.*, (17):211–227, 1993.

[Kal02] J. Kallrath. Combined strategic and operational planning - an milp success story in chem. ind. *OR Spectrum*, (24):315–341, 2002.

[SPE04a] O. Stursberg S. Panek and S. Engell. Job-shop scheduling by combining reachability analysis with linear programming. In *7th Int. IFAC Workshop on Discrete Event Systems*, pages 199–204, 2004.

[SPE04b] O. Stursberg S. Panek and S. Engell. Optimization of timed automata models using mixed-integer programming. In *Formal Modeling And Analysis of Timed Systems*, volume volume 2791 of *LNCS*, pages 73–87. Springer, 2004.

[SPE06] O. Stursberg S. Panek and S. Engell. Efficient synthesis of production schedules by optimization of timed automata. *Control Engineering Practice*, 14(10):1183–1197, 2006.

[SPS] S. Engell S. Panek and O. Stursberg. Scheduling and planning with timed automata. In *16th Europ. Symp. on Computer-Aided Process Engineering*.