CHAPTER 3

REAKTION: a system for event independent reactive scheduling

H. Henseler
Universität Oldenburg, Fachbereich Informatik, PF 2503, D-26111 Oldenburg, Germany

Abstract
This paper proposes a new concept for reactive scheduling in detailed schedules which have become inconsistent due to events. The events can be of external type, e.g. short-term acceptance of a high-priority order or delay of material delivery, or of internal type, e.g. a machine breakdown. After formalizing the general scheduling problem and a reactive scheduling model we will present an algorithm which efficiently "repairs" the violated constraints by revising the last schedule. We will show that this repair process can be divided into two parts: One is event-specific and the other is a method for "general repair". A look at the implementation in Prolog closes the paper.

Keyword Codes: F.2.2; G.1.6; I.2.1
Keywords: Sequencing and Scheduling; Optimization; Industrial Automation

1. SCHEDULING

Job-shop scheduling as a means of assigning resources in flexible manufacturing has grown to a main tool in the logistic area of enterprises. In different steps, ranging from master schedules, material requirements planning and production planning to resource scheduling, plans are getting more and more detailed. Finally the time and resource required for every step of every job is fixed. This final schedule is the frame for the production in the plant. However, it is usually not respected that deviations from this schedule and disturbances in the plant can and will occur. Insufficient ad-hoc solutions, which put the results of the planning process in question, are used commonly. A production planning and control system (PPC) creates a set of orders assigned with start times and end times for every order. The purpose of scheduling is to create a detailed schedule from this results which respects technical constraints and economical goals. Production of an order is accomplished by using resources, usually machines and employees. The production plan describes which operations have to be executed for how long and on which machine(s). The schedule fixes the exact start time, duration and resource for every operation of every order. The goal "economical" means to satisfy global goals of the enterprise, mainly minimizing costs. A way to ensure this is e.g. to minimize the work-in-process time without creating tardy orders. Such goals are conflicting in nature and increase the difficulty of the scheduling problem [1].

© IFIP International Federation for Information Processing 1995

1.1. A model for the scheduling problem

The quadruple $GSP = (M, T, P, O)$ is called model for the *general scheduling problem*, where M, T, P and O are defined as follows:

1. $M = \{M_1, \ldots, M_m\}$ is a set of *machines*. These can be actual machines found on the shop floor, workers or transport units.
2. $T = \{T_1, \ldots, T_m\}$ is a set of *time axes*, $T_i \subseteq Z$, where Z is the set of all *time units* of a discrete time axis. A time axis $T_i \in T$ represents which time units on a machine M_i are work times and which are down times. Work times for machine M_i are those time units which are element of the time axis T_i. The real size of time units can be selected application dependent, like minutes, hours or days.
3. $P = \{P_1, \ldots, P_p\}$ is a set of *products*. Every product $P_i = \{V_{i_1}, \ldots, V_{i_k}\}$ is defined by a set of production *variants* (different methods of producing the same type of order), where every variant $V_{i_j} = (\{Op_{i_{j_1}}, \ldots, Op_{i_{j_y}}\}, <_{i_j})$ consists of a set of *operations* and a binary *operation precedence relation* on these operations. This relation defines a successor- and predecessor-relation on the set of operations of a variant, i. e. $<_{i_j}$: $Op \times Op$. Every operation $Op_{i_{j_k}}$ is given a set of *alternative machines* and corresponding *execution times* $\{(M_{i_{j_{k_1}}}, a_{i_{j_{k_1}}}), \ldots, (M_{i_{j_{k_d}}}, a_{i_{j_{k_d}}})\}$, with $M_{i_{j_{k_x}}} \in M$ and $a_{i_{j_{k_x}}} \in N, 1 \leq x \leq d$. The execution time of an operation is the number of time units needed to execute the operation on this machine.
4. $O = O_1, \ldots, O_n$ is a set of *orders*, $O_j = (P_j, s_j, e_j, p_j)$. $P_j \in P$ is the product which has to be produced, $s_j \in Z$ the predefined start time, $e_j \in Z$ the predefined end time for production, and $p_j \in N$ a priority value. The start time is given as the earliest *material availability*, the end time usually marks the *delivery* of the product to the customer. The priority expresses the economical relevance of the order.

Note that there is no parameter which describes what amount of a product has to be produced. This is expressed by different products with different execution times. Also there is no modelling of cleaning times between operations on one machine. This is also expressed in the product description and therefore does not depend on operation ordering.

1.2. A model for the schedule

Given a scheduling problem $GSP = (M, T, P, O)$. A scheduling problem solution (short: *schedule*) $SPS = (o_1, \ldots, o_n)$ relative to GSP is a set of *scheduled orders*. For every element $O_i \in O$ we assign an o_i with the property: $o_i = (o_{i_1}, \ldots, o_{i_o})$ is a set of *scheduled operations*. Every $o_{i_j} = (M_{i_j}, s_{i_j}, e_{i_j})$ is a triple consisting of a machine $M_{i_j} \in M$, a start time $s_{i_j} \in Z$ and an end time $e_{i_j} \in Z$. As a result, a schedule assigns every order a sequence of operations with exact start- and end times on a particular machine. Schedules are visually represented with so called gantt-charts (see fig. 5).

1.3. Constraints between the scheduling problem and a schedule

There are two kinds of constraints between a scheduling problem and the schedule: On one hand those which *must* be respected by a schedule (*Strong constraints*), on the other hand those which *should* be respected by a schedule (*soft constraints*). The first guarantee the technical correctness and completeness of a schedule, the latter the economical quality.

Strong constraints are

C_1 All operations must be produced on correct machines (defined by the correspon-ing product). All alternative machines are equivalent in terms of exchangeability. Note that the execution times may differ.

C_2 All orders are assigned to the correct operations, corresponding to one variant.

C_3 Every order is manufactured by executing one variant of the corresponding product.

C_4 All orders have to be executed.

C_5 The start time of an operation must not be smaller than the start time of its order.

C_6 Non-working time units on machines do not count for the execution time of operations.

C_7 The execution time of an operation $Op_{i_{j_k}}$ on a machine $M_{i_{j_{k_l}}}$ corresponds to the time fixed by $a_{i_{j_{k_l}}}$. Every operation has to be executed without interruption. An operation may cross non-working time units of a machine.

C_8 For every time unit there is only one operation on every machine.

C_9 Orders have to respect the operation precedence relation of their variant.

Soft constraints are

C_e The end time of an operation should not be greater than the end time of its order.

C_o The plan should be optimal in respect to some evaluation function.

A schedule SPS is called a *solution* of a scheduling problem GSP if it respects all strong constraints.

It is important to note that soft constraints can not be expressed by a simple formula. They are one source for "knowledge" which has to be taken into account. The person who creates a schedule by hand often has only a vague idea of the rules which guide his scheduling decisions. Terms like "low tardiness", "short work-in-process times", or "good utility of critical machines" may be used to express his knowledge [2].

2. REPLANNING

The reality in the factory is characterized by many short-term changes of the schedule. This may happen due to a variety of reasons. Since the execution times of operations are often based on assumptions or data of former experience or are simply guessed, the real execution time will certainly deviate from the scheduled execution time. Moreover, technical problems, e.g. machine breakdowns or staff shortage can make a *schedule correction* necessary. Such disturbances, which are based in the production process are called *internal disturbances.*

The other form is called *external disturbances* and covers all changes of the frame for a schedule. For example an order cancellation may change the number of orders, or an additiona high-priority order arrives and must be scheduled within strong time bounds. The simplest way is to throw away the old schedule and create a totally new one. But this solution is not satisfactory due to the following reasons:

- disturbances and derivations tend to appear very often
- the effort to create a new schedule is usually very high
- preparations of operations (material orders, material transport, tool changes, cleaning, etc.) may become invalid although the disturbance has nothing to do with them.

First, devising an initial schedule does not take into account the actual floor shop situation. Instead, the schedule will be created for a fixed period of time, e.g. one week and after this period all leftover work and work for the next week is used to generate the next schedule. This aproach is insensitive to greater disturbances like a machine breakdown. Second, observations of production processes have shown that many disturbances can be repaired with small changes to the schedule. Small deviations from start times or end times are normal. In fact, although the production may differ slightly from the schedule, it is not updated because the difference seems to be so minimal. On the other hand there is no guarantee that these deviations do not accumulate or create conflicting situations. Both problems are solved by using large buffers between individual operations of orders. This acts against these problems, but it increases work-in-process time whereas small buffers increase the need of reactive scheduling. Moreover, scheduling very rarely starts at zero, i.e. without any preconditions of machine load, disposition of material, etc., which adds an even greater demand for a reactive method.

The following actions are suitable to face disturbances:

- time adjustment (e.g. short-time work or overtime hours)
- intensity adjustment (e.g. higher or lower running speed of machines)
- quantitative adjustment (shut down machines or starting formerly used machines)
- lot size adjustment (splitting or combining) orders
- outward manufacturing of operations or orders
- time adjustments of orders.

Solving disturbances by picking one action without adjusting the schedule does not consider existing dependencies expressed by constraints. But this is the focus problem when dealing with reactive scheduling [3].

In the following we will define the replanning problem as finding a new Solution SPS' for a slightly changed scheduling problem GSP' which is derived from the problem GSP with an already existing solution SPS.

When the schedule SPS is set into reality and an event occurs, part of the schedule already has become reality and can not be changed. The distinction between irreversible and changeable parts of a solution will be made with a special time unit N (= *now-unit*) which represents the time unit where the event occurs on the time axis Z. The state of a machine in a time unit smaller N is history and therefore unchangeable, whereas states in time units bigger or equal to N are future and therefore changeable.

The *machine-state* of a machine $M_i \in M$ in a solution SPS of the scheduling problem $GSP = (M, T, P, O)$ at a time unit $z \in Z$ is

- O_k, when machine M_i produces an operation for the order $O_k \in O$ in time unit z
- "empty", when $z \in T_i$ and no operation is scheduled for M_i in z
- "non-working", when $z \notin T_i$.

2.1. A model for the reactive scheduling problem

A model for the *reactive scheduling problem* is given by $RSP = (GSP, SPS, D)$ where:

1. $GSP = (M, T, P, O)$ is a scheduling problem.
2. SPS is a solution for GSP.
3. $D = (E, N)$ is a *disturbance*, E an *event* and $N \in Z$ is a time unit at which E occurs.

An event E is a change of a scheduling problem. As a result, not only external and internal disturbances are events, but also actions.

Table 1
Overview events and possible inconsistencies

#	event	change	N	1	2	3	4	5	6	7	8	9	e	o
	concerning order:													
1	new order	O					×							
2	cancellation of order	O		×			×							
3	change of start time	s_i		×				×						
4	change of end time	e_i											×	×
5	change of amount	P_i	×			×				×				
6	change of priority	p_i												×
7	change of product	P_i/V_i	×	×	×	×				×				
8	order splitting	O, P_i	×			×	×			×				
	concerning machine:													
9	disturbance / repair	T_i							×	×				
10	regular repair period	O					×							
11	machine intensity	P_i	×			×				×				
12	change of shifts	T_i	×						×	×				
13	number of machines	T_i	×						×	×				
	concerning operation:													
14	new shop floor data	P_i				×				×				
15	outward production	P_i	×	×	×	×								

The set $E = \{E_1, \ldots, E_{15}\}$ is a set of event types, defined as follows (see table 1): The column "change" shows which values or sets in GSP and GSP' will become different due to the event. The column "N" to "o" show which constraints can be violated (marked with "×").

To express the irreversibility of machine states in the past we define a strong constraint for two solutions SPS and SPS' and a time unit N:

C_N The machine-states of all machines for all time units $z \in Z, z < N$ in SPS and SPS' are identical.

2.2. Solution of a reactive scheduling problem

Given a reactive scheduling problem $RSP = (GSP, SPS, D)$ with disturbance $D = (E, N)$ and a scheduling problem $GSP = (M, T, P, O)$. The tuple (GPS', SPS') is the called *solution of the reactive scheduling problem RSP*, when the following holds:

1. $GSP' = (M, T', P, O')$ is a scheduling problem which results from applying the event E to P.
2. SPS' is a solution for GSP'.
3. SPS and SPS' are consistent relative to C_N for time unit N.

If we look at the tuples (GSP, SPS) and (GSP', SPS') as consistent states, an event E which changes SPS can be considered as a beginning of a *transaction*. Then we have to find a series of reactions $R_0, \ldots, R_n$ which, beginning with the event E, transform (GSP, SPS) into the consistent state (GSP', SPS'). The next problem is to find out how such reactions might work and how to find an SPS' for a given RSP.

3. AN ALGORITHM FOR REACTIVE SCHEDULING

We now present an algorithm which solves the reactive scheduling problem.

A *reaction* is a (maybe empty) series of *basic operations* which transpose a tuple (GSP', SPS_i) into a tuple (GSP', SPS_{i+1}). Basic operations are creating and deleting operations and changing of operation data (machine, start time, end time).

The algorithm will proceed as follows: The event E transposes the scheduling problem GSP into GSP'. Since SPS will be no longer a solution for GSP', a reaction R_0 is necessary which tries to repair the inconsistency. Usually, applying only one reaction will not solve the inconsistency because the inconsistency is not fully repaired or new inconsistencies will be introduced by R_0. Therefore subsequent reactions $R_1, R_2, \ldots$ are necessary. The following scheme demonstrates the flow of this process ($X - Y \rightarrow Z$ means "X becomes transposed into Z with event or reaction Y")

$$\begin{array}{lll}
(GSP, SPS) - E \rightarrow (GSP', SPS) & & \\
\quad = (GSP', SPS_0) & -R_0 \rightarrow & (GSP', SPS_1) \\
\quad (GSP', SPS_1) & -R_1 \rightarrow & (GSP', SPS_2) \\
 & \vdots & \\
\quad (GSP', SPS_n) & -R_n \rightarrow & (GSP', SPS_{n+1}) = \\
 & & (GSP', SPS').
\end{array}$$

The result is a process shown in fig. 1. Scheduling problem and solution are "chained" together by constraints. While E changes the problem (and breaks the chain) the reactions change the solution to fix the chain.

A reaction R_i results from the previous changes $E, R_0, \ldots, R_{i-1}$ and (GPS, SPS), i.e. it is not derived from the tuple (GSP', SPS_i). Because of this reactions are always

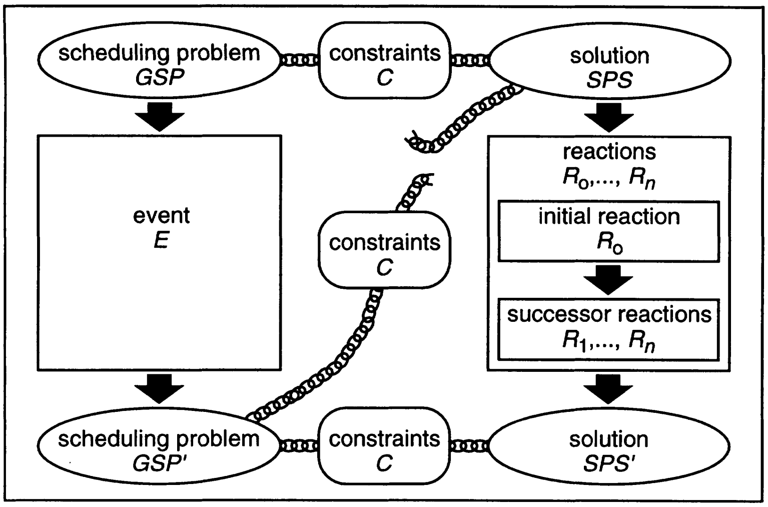


Figure 1. How to achieve consistency

postulated. That means when the tuple is changed, care is taken to new inconsistencies. These inconsistencies are postulated as reactions which have to be executed in one of the next steps. This is more efficient than always searching for all inconsistencies in the current tuple. Since a change can result in many necessary reactions, these have to be collected. Lets call the set of collected reactions PR ("postulated reactions"). As a result, reactions have a dual nature: On the one hand they are procedures which repair an inconsistency, on the other hand they represent an inconsistency.

Formally this means: $\forall i \in N_0, 0 \leq i \leq n+1 : PR_i = \{R_{i_1}, R_{i_2}, \ldots, R_{i_m}\}$.

Conditions for postulated reactions are:

1. PR_0 only depends on E.

2. Every postulated reaction in PR_i is being executed: $\forall i \in N_0, 0 \leq i \leq n, \forall R \in PR_i, \exists k, i \leq k \leq n : (GSP', SPS_k) - R \rightarrow (GSP', SPS_{k+1})$ and $R \notin PR_{k+1}$.

3. $PR_{n+1} = \emptyset$.

The sets PR_i represent the inconsistencies in the tuple (GSP', SPS_i) which still have to be repaired. They consist of the postulated but not yet executed reactions. The algorithm will execute reactions from PR_i until the set PR_i becomes empty for one i. Then the tuple (GSP', SPS_i) will be consistent and a solution of the reactive scheduling problem.

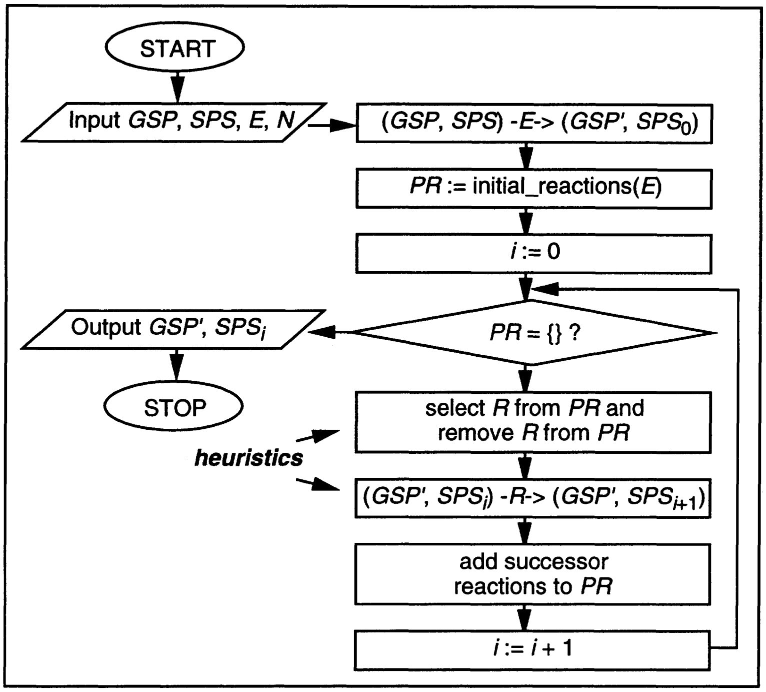

START
Input GSP, SPS, E, N
(GSP, SPS) -E-> (GSP', SPS0)
PR := initial_reactions(E)
i := 0
Output GSP', SPSi
PR = {} ?
STOP
select R from PR and remove R from PR
heuristics
(GSP', SPSi) -R-> (GSP', SPSi+1)
add successor reactions to PR
i := i + 1

Figure 2. Basic algorithm

Figure 2 shows the flowchart of the frame of the reactive scheduling algorithm (compare [4, 5]). There are two points where the algorithm needs the help of specific knowledge to reach results of good quality (and to reach a result at all):

- determine which reaction in PR has to be executed next
- how should the reactions proceed.

This knowledge may be adopted from a human scheduler. By learning his method of rescheduling one may find good tactics for this two questions. For example, he might prefer to solve inconsistencies which belong to bottleneck machines first. Or he might look at operations on other machines before he resolves inconsistencies concerning operation order. We benchmarked different approaches of reaction ordering and found that a strategy which tries to repair the schedule in increasing order of time, beginning with time unit N produced the best results both for rescheduling time used and for producing the best schedules.

According to table 1 an event can break many constraints. We will now classify constraints in terms of their reactive scheduling behaviour. The goal is to get hints on how reactions should proceed. What consistencies should they break in order to repair the inconsistency which lead to their own postulation? Here we use a "trick": operations which will overlap with other operations due to their addition or change of length will be temporarily removed from the "real" machine and placed on a virtual *buffer machine.* Operations on buffer machines will not break any of the constraints C_1 to C_8. However, to reach an overall consistent plan such operations must be brought back to the real machine during the reactive scheduling process, perhaps by moving other operations to the buffer machine. Therefore a new constraint has to be defined:

C_B No operation is placed on a buffer machine.

Table 2
Classes of consistency

consistency class	concerned constraints	reactive scheduling behaviour	caused by
CC_I	C_N, C_2, C_8	-	never
CC_{II}	$C_1, C_3 - C_7$	simple (one initial reaction)	event
CC_{III}	C_9, C_B, C_e, C_o	difficult (many successor reactions)	event, reaction

The classification scheme is outlined in table 2. We can now divide the process of reactive scheduling into processing an *initial reaction* and some *successor reactions.* The initial reaction depends only on the event and simply places all operations which violate any constraint to a buffer machine. The successor reactions depend only on the content of the set *PR.* This distinction allows to add new events to the algorithm only by providing a new initial reaction. Additionally, it allows to view reactive scheduling independent of the occurred events. Now one part of the algorithm contains handling of events, the other part contains the algorithmic effort to reach consistency. The latter also contains all reactive scheduling knowledge.

Now a reaction is reduced to deciding what change must be made to repair the inconsistency which caused its postulation. The following global conditions have to be taken into account for this decision:

1. Only constraints from class CC_{III} should be violated.

2. The algorithm must terminate.

3. The algorithm should find a "good" solution considered to C_o and C_e.

Condition 1 is guaranteed by using the buffer plan whenever overlapping of operations is unavoidable. The conflicting operation is simply moved to the buffer plan and violates C_B. How the termination problem is treated is shown in fig. 3. The diagram outlines what constraints can be violated when another constraint is repaired by a reaction. There are three cases which are of special interest for the termination problem because of "postulation-loops":

- between C_9 and C_B: A precedence conflict is solved by moving the successor operation to the buffer plan. When moving this operation back to the real plan a new precedence conflict can occur. This case will not harm the termination of the algorithm because of the linear order on the set of operations.

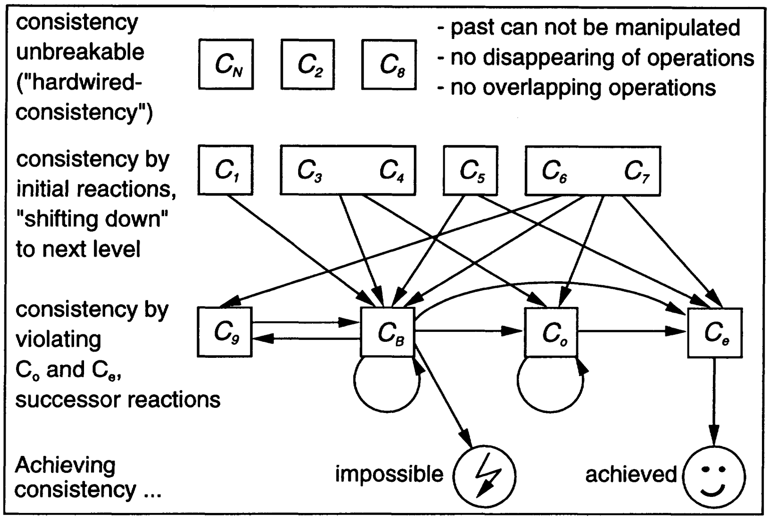

consistency unbreakable ("hardwired-consistency")
C_N
C_2
C_8
- past can not be manipulated
- no disappearing of operations
- no overlapping operations
consistency by initial reactions, "shifting down" to next level
C_1
C_3 C_4
C_5
C_6 C_7
consistency by violating C_o and C_e, successor reactions
C_9
C_B
C_o
C_e
Achieving consistency ...
impossible
achieved

Figure 3. Levels of achieving consistency

- between C_B and C_B: Moving an operation back to the plan may cause an other operation to move to the buffer plan. One solution to the problem is to use a precedence relation on the operations. For example an operation may only replace operations from an order with higher priority. Another solution is to use time stamps which inhibits operations to fall back again to the buffer plan once they are back in the real plan.

- between C_o and C_o: We understand repairing C_o as filling gaps which are caused by moving operations. For this purpose we shift operations left to fill out the gap. This may produce other gaps, but they concern time units which are bigger than those from the original gap. Repairing C_o can be considered as a "shift left" of operations.

Note that there are situations where the algorithm is not able to find a solution although it may exist. Whenever a machine breaks down completely and alternative machines are filled with operations of higher priority, the algorithm has no chance to select the right strategy. Luckily, this scenario is very artificial and usually does not occur.

There is still one question left: How should the reactions proceed in order to produce good schedules? Once again, the algorithm needs some sort of knowledge, which depends on the factory floor, the number and type of typical events and on the current goal: Fast or good rescheduling. Interestingly, only two reactions really need this knowledge: First, the reaction to solve C_B must know where to put the operation from the buffer machine back to the schedule. Should it push other operations to the buffer plan or try to find a fitting gap? And second, the reaction for C_o must know how to fill gaps. Which operations should be pushed left?

We gathered obvious rescheduling knowledge concerning priority, gaps and bottleneck machine handling and made it available to the algorithm in form of heuristics and rules.

Examples:

- "If an operation is moved to a bottleneck machine, try to fill the first possible gap"
- "If an operation is moved to a non-bottleneck machine and the priority of the order is low, try to find a best fitting gap".

Tests with different shop floors showed that balancing heuristics is difficult and that there is no general rescheduling strategy which produces good results in all shop floors.

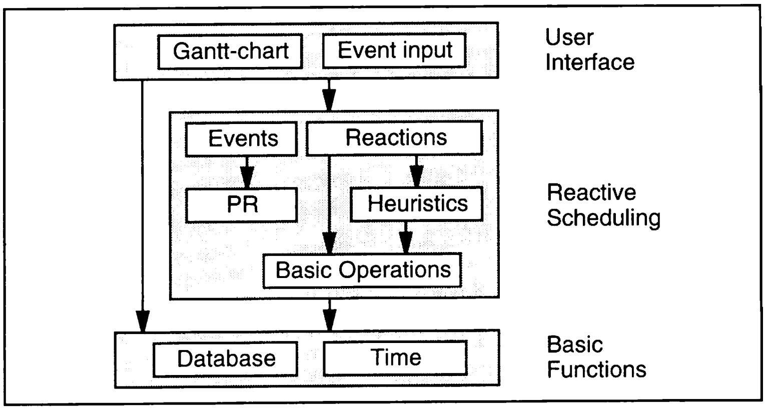
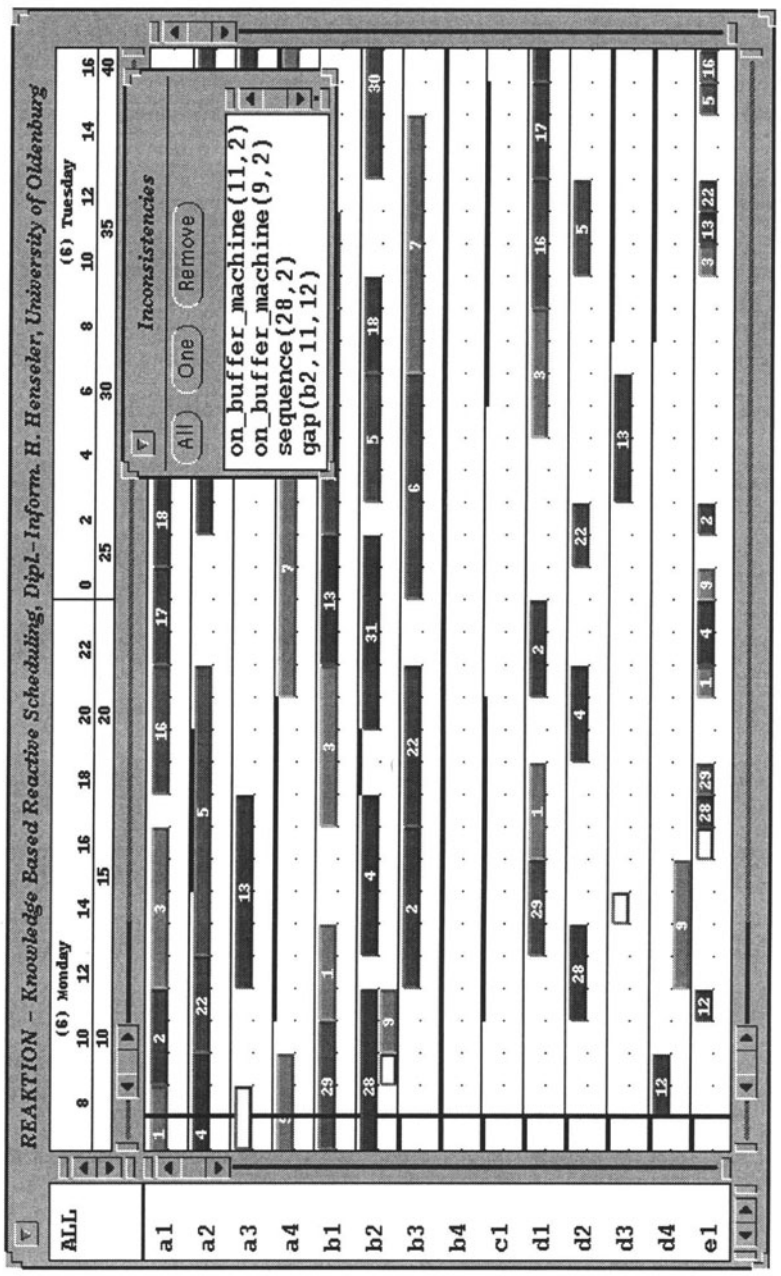


Figure 4. Architecture of REAKTION

4. THE SYSTEM REAKTION

The proposed algorithm is implemented in a system called REAKTION (acronym for a German description). It contains the automatic and manual support of restoring consistency of schedules. The system was implemented in Quintus Prolog on a Sun SparcStation. A special focus was aimed at integrating a graphical user interface for manual reactive scheduling.

The structure of REAKTION is depicted in fig. 4: The system is divided into three modules: basic functions, reactive scheduling and user interface. The module basic functions contains the database and maintenance of time axes. The module reactive scheduling manages the heuristics to guide the reactive scheduling process, the set *PR*, reactions and events. The user interface interactively shows the schedule in form of a gantt-chart and reacts to user input. The human scheduler has the opportunity to move operations via a mouse. These movements are also treated as events which influence the schedule. Results of starting the algorithm for automatic reactive scheduling are directly visible on the screen. The user has direct influence on the set *PR* which is visible in an extra window. A screendump is shown in fig. 5.


REAKTION - Knowledge Based Reactive Scheduling, Dipl.-Inform. H. Henseler, University of Oldenburg
ALL
(6) Monday
(6) Tuesday
Inconsistencies
All
One
Remove
a1
a2
a3
a4
b1
b2
b3
b4
c1
d1
d2
d3
d4
e1

```
on_buffer_machine(11,2)
on_buffer_machine(9,2)
sequence(28,2)
gap(b2,11,12)
```

Figure 5. Screendump of REAKTION

A connection to the system PSY [6], developed as a scheduling tool for metal working industry, demonstrated the utility of REAKTION for an existing plant.

5. CONCLUSION

In this paper we presented an algorithm for reactive scheduling. Highlights of this algorithm are the ability to handle all possible events which may happen in a shop floor and incorporation of arbitrary heuristics. It is expandable in a way that new events can be added at any time without need to change the algorithm itself. The distinction between reacting and repairing seems promising for future research. As the buffer machine concept seems to be a natural way of resolving constraint conflicts the integration of the user in the rescheduling cycle is very elegant. The human scheduler may act on critical decisions by himself and let the algorithm do the rest of the work. It is possible to stop the algorithm at any time to make adjustments by hand or to integrate new events. The algorithm uses an opportunistic search, i.e. it does not explore the whole search tree but focuses on only one, carefully choosen path. This makes the rescheduling cycle very fast.

There are nevertheless disadvantages which have to be taken care of. After finding a new schedule the algorithm stops and therefore misses a chance to improve schedule quality further. And as the result of the opportunistic search, the algorithm is not symmetric in that deleting the effect of a previous event does not lead back to the previous schedule.

We are currently addressing new methods of rescheduling, namely distibuting the schedule. Some of our new ideas can be found in [7].

REFERENCES

1. H. Parunak. Characterizing the Manufacturing Scheduling Problem. Journal of Manufacturing Systems. Vol. 10, No. 3, 1991.

2. M. Fox. Constraint Directed Search: A Case Study of Job-Shop Scheduling. Pitman Publishers, London, 1987.

3. K. G. Kempf. Manufacturing Planning and Scheduling: Where We Are and Where We Need To Be. Proceedings CAIA-89, Miami, FL, 1989.

4. P. S. Ow, S. F. Smith, A. Thiriez. Reactive Plan Revision. AAAI 88.

5. F. S. Smith, P. S. Ow, D. C. Matthys, J. Y. Potvin. OPIS: An Opportunistic Factory Scheduling System. Carnegie-Mellon University. Pittsburgh, U. S. A., 1989.

6. J. Sauer, R. Bruns. Knowledge-Based Scheduling Systems in Prolog. International Conference on The Practical Application of PROLOG, 1994.

7. H. Henseler. From Reactive to Active Scheduling. In this book.