A SURVEY OF INTERVAL CAPACITY CONSISTENCY TESTS FOR TIME- AND RESOURCE-CONSTRAINED SCHEDULING

Ulrich Dorndorf, Toàn Phan Huy, Erwin Pesch[*]

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Gesellschafts- und Wirtschaftswissenschaften, BWL III
Adenauerallee 24-42, D-53113 Bonn, Germany
udorndorf@acm.org

Abstract: Interval capacity consistency tests consider the resource capacities available and required within certain time intervals. The goal of the tests is to draw conclusions that allow to rule out inadmissible activity start times or sequences. The tests can be effectively used to reduce the search space of difficult time- and resource-constrained scheduling problems. They have successfully been applied in algorithms for solving idealised problems such as the classical job shop scheduling problem (JSP) or the resource-constrained project scheduling problem (RCPSP) as well as for solving industrial scheduling problems. For instance, it seems fair to say that the advances in modern branch and bound algorithms for the JSP that have been made in the last decade can to a large extent be attributed to the effect of interval consistency tests, some of which are also known under the names of immediate selection, edge finding, and energetic reasoning. The tests can also serve to derive tight lower bounds for makespan minimisation problems.

This chapter presents interval consistency tests for disjunctive and general resource-constrained (cumulative) scheduling within a unified framework, using numerous examples for illustration. We review the state of the art and derive some new results for disjunctive scheduling.

1 Introduction

Time and resource constrained scheduling is concerned with the task of scheduling a number of activities subject to temporal constraints between activities, such as

[*]To appear in: *Handbook on Recent Advances in Project Scheduling,* J. Węglarz, ed. Kluwer Academic Publishers, Boston, Dordrecht, London, 1998.

precedence or synchronisation, and constraints on the availability of several shared resources. An activity i is characterised by its processing time p_i and resource requirements r_{ik}: It requires r_{ik} units of a renewable resource k for each of p_i time units, and it releases the resource units again upon completion. Once begun, an activity must be processed in an uninterrupted fashion. A resource k is available in constant amount R_k.

By imposing an upper bound on the latest completion time and a lower bound on the earliest start of all activities, activity start time windows can be derived in a straightforward way. The start time window of activity i is the interval between the earliest start time est_i and the latest start time lst_i of i. The *domain* δ_i is the set of all possible start time assignments st_i of i. It is bounded by the start time window; because some values in the start time window may be excluded, e.g. by temporal constraints of the type $st_i \neq t_x$, for some time t_x, we can state in general that $\delta_i \subseteq [est_i, lst_i]$.

The purpose of this chapter is to present a class of logical tests called *interval capacity consistency* tests which are based on resource constraints. These tests allow to reduce activity domains by ruling out infeasible start time assignments. They can be applied in scheduling algorithms such as list scheduling or branch and bound procedures, or in constraint propagation based scheduling systems. The benefit of the tests is that they can reduce the search space and direct an algorithm towards good solutions. Here, we are only interested in the tests themselves and do not address scheduling algorithms in which they can be embedded. Since the tests only eliminate solutions incompatible with the capacity constraints, they are independent of the overall objective function to be optimised. The assumptions that we have made so far are rather general and cover $\mathcal{NP}$-hard models such as the classical job shop problem (JSP) (Błażewicz et al. 1996a,b) and the resource-constrained project scheduling problem with simple (RCPSP) or generalised precedence constraints (RCPSP/max) (Brucker et al. 1998, Domschke and Drexl 1991). In models with generalised precedence constraints time varying resource supply can easily be reflected by introducing dummy activities (Bartusch et al. 1988).

In the following sections we assume that all activity domains have been made consistent with the temporal constraints. Efficient algorithms for this are well known; a detailed description of this process in a constraint propagation based scheduling system is for instance given by Nuijten (1994). We are interested in further reducing the domains by applying interval consistency tests.

In the literature, activity domains are often approximated by start time windows, and this approximation is then referred to as activity release times and due dates, or heads and tails. The domain reduction process may then be called adjustment of heads and tails or time bound adjustment. Specific interval consistency tests have become known under the names immediate selection, edge finding, and energetic reasoning.

The remainder of this chapter is organised as follows. Section 2 introduces some additional formal notation and a graphical representation used for the examples throughout the chapter. Section 3 discusses the concept of interval consistency and shows the general idea how to deduce domain reductions. Section 4 then goes into more detail and takes a bottom up approach to derive the tests for the case of unit resource capacities and requirements (*disjunctive* scheduling). Section 5 generalises the results for arbitrary resource capacities and requirements (*cumulative* scheduling).

2 Notation

Let us first briefly introduce some additional notation. We denote the set of all activities as $\mathcal{V}$ and the subset of all activities to be processed by resource k as $\mathcal{V}_k \subseteq \mathcal{V}$, with $\mathcal{V}_k = \{i \in \mathcal{V} | r_{ik} > 0\}$. We will often consider subsets $\mathcal{A} \subseteq \mathcal{V}_k$ of activities. To deduce domain reductions for an activity i and activities in a set $\mathcal{A}$ we often try to show that i must start before or finish after all activities in $\mathcal{A}$. This is denoted by $i \rightarrow \mathcal{A}$ if i must start first, and $\mathcal{A} \rightarrow i$ if i must finish last. We also use the notation $\mathcal{A} \rightarrow \mathcal{A}'$, with $\mathcal{A}, \mathcal{A}' \subseteq \mathcal{V}_k$ to express that all activities in set $\mathcal{A}$ must start before all activities in set $\mathcal{A}'$. Table 1 summarises the notation used throughout this chapter.

For illustration and motivation of the tests, we use examples in the style of Figure 1, which shows two activities that must be processed by the same resource with capacity $R_k = 1$. The style is similar to the one used by Nuijten (1994). Consider activity

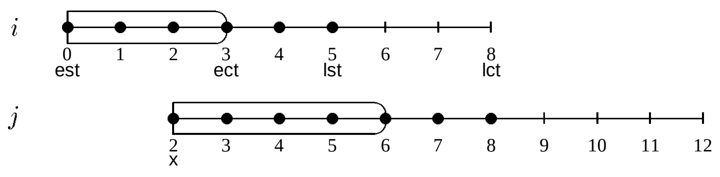

Figure 1 Two activities $i \in \mathcal{V}_k$ and $j \in \mathcal{V}_k$ with $p_i = 3$ and $p_j = 4$

i where several points on the time scale have been annotated for illustration. The figure shows the time between the earliest start of i, est_i, and its latest completion, lct_i, as a horizontal line segment. For convenience, the processing time of i, p_i, is depicted as a hollow bar beginning at est_i with rounded right end at $ect_i = est_i + p_i$; the length of this bar is, of course, equal to lct_i - lst_i. Admissible start times, i.e. the values in δ_i, are shown as black circles. Times in the interval $[lst_i + 1, lct_i)$ at which i may be in process, but at which it cannot start, are marked with tick marks. You can think of scheduling an activity as positioning the processing time bar at one of the admissible start times. Activity j in Figure 1 appears in the usual style without annotations. Initially, possible start times of j are in the interval $[2, 8]$. The x appearing under j's scale at time 2 indicates that we have — by applying a test described below — deduced that j cannot start at time 2.

3 Interval consistency

Before looking at the individual consistency tests, this section introduces a general framework for the tests.

Symbol	*Description*
δ_i	domain of activity i: the set of possible start times of i
est_i	earliest start time of activity i
ect_i	earliest completion time of activity i
lst_i	latest start time of activity i
lct_i	latest completion time of activity i
p_i	processing time of activity i
$p_i(t_1, t_2)$	smallest amount of time during which activity i must be processed in time interval $[t_1, t_2)$
r_{ik}	amount of resource k required by activity i
st_i	actual start time of activity i
t	time period
$w_{ik}(t_1, t_2)$	smallest amount of work of activity i that must fall into the time interval $[t_1, t_2)$: $r_{ik}p_i(t_1, t_2)$
$\mathcal{A}$	set of activities to be processed by resource k: $\mathcal{A} \subseteq \mathcal{V}_k$
$Ect(\mathcal{A})$	preemptive bound for the earliest completion time of activity set $\mathcal{A}$
$\mathcal{H}$	set of hypothetical constraints
$Lst(\mathcal{A})$	preemptive bound for the latest start time of activity set $\mathcal{A}$
$P(\mathcal{A})$	total processing time of activities in set $\mathcal{A}$: $\sum_{i \in \mathcal{A}} p_i$
$P(\mathcal{A}, t_1, t_2)$	interval processing time of $\mathcal{A}$: $\sum_{i \in \mathcal{A}} p_i(t_1, t_2)$
R_k	constant capacity of resource k
$\mathcal{V}$	set of all activities
$\mathcal{V}_k$	set of activities requiring resource k: $\{i \in \mathcal{V} \| r_{ik} > 0\}$
$\mathcal{V}_k(t_1, t_2)$	set of activities requiring resource k that must be completely or partially processed within $[t_1, t_2)$: $\{i \in \mathcal{V}_k \| p_i(t_1, t_2) > 0\}$
$W(\mathcal{A})$	total work of activities in set $\mathcal{A}$: $\sum_{i \in \mathcal{A}} w_i$
$W(\mathcal{A}, t_1, t_2)$	interval work time of activities in set $\mathcal{A}$: $\sum_{i \in \mathcal{A}} w_{ik}(t_1, t_2)$

Table 1 Summary of notation

An activity i requires an amount of work $w_i = r_{ik}p_i$ from resource k. We say a time interval is capacity consistent if the amount of work available from resource k within the interval is greater than the interval work required by all activities. Let us consider the work of an activity that must fall into a time interval $[t_1, t_2)$. Figure 2 shows the four possible relations between an activity and a given time interval where the activity must be completely or partially processed within the interval. The activity can be (1) completely contained within the interval, (2) completely overlap the interval when started as early (left-shifted) or as late (right-shifted) as possible, (3) have a minimum processing time within the interval that is realised when started as early as possible, or (4) have a minimum processing within the interval that is realised when started as late as possible. The smallest amount of time, or *interval processing time,* during which

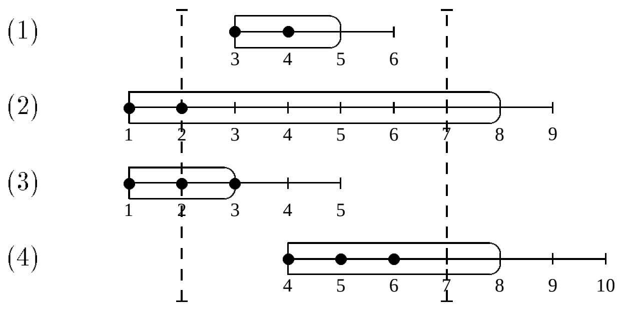

Figure 2 Types of intersections between activities and a time interval

an activity i must be executed in $[t_1, t_2)$ is

$$p_i(t_1,t_2) = \max\left\{0, \min\left\{p_i, t_2 - t_1, ect_i - t_1, t_2 - lst_i\right\}\right\}, \tag{1}$$

and the corresponding *interval work* is $w_{ik}(t_1,t_2) = r_{ik}p_i(t_1,t_2)$. The interval processing time for a set $\mathcal{A} \subseteq \mathcal{V}_k$ is $P(\mathcal{A}, t_1, t_2) = \sum_{i \in \mathcal{A}} p_i(t_1,t_2)$; the corresponding work within the interval is $W(\mathcal{A}, t_1, t_2) = \sum_{i \in \mathcal{A}} w_{ik}(t_1,t_2)$. Inversely, we denote the set of all activities that must be processed completely or partially within $[t_1, t_2)$ as $\mathcal{V}_k(t_1,t_2) = \{i \in \mathcal{V}_k | p_i(t_1,t_2) > 0\}$.

Using the interval work definition we can now state an interval consistency constraint in its general form:

$$R_k \cdot (t_1 - t_2) \geq W(\mathcal{V}_k, t_1, t_2). \tag{2}$$

Clearly, a scheduling problem can only have a feasible solution if this constraint holds for all resources k and time intervals $[t_1, t_2)$. However, this is only a necessary condition and does not guarantee that a feasible schedule exists.

The basic idea behind all interval consistency tests described in this chapter is as follows: We consider a set of additional, hypothetical constraints $\mathcal{H}$ and try to show that if $\mathcal{H}$ is satisfied then Constraint 2 is violated for some resource and time interval; in this case we can conclude: $\neg\mathcal{H}$. This leads to two main questions which we will try to answer in the following sections:

1. How should we choose $\mathcal{H}$ so that the conclusion $\neg\mathcal{H}$ leads to useful domain reductions?

2. For which intervals $[t_1, t_2)$ should we test Constraint 2?

Tests are usually applied in an iterative fashion for as long as some domain can be reduced.

The notion of interval capacity consistency as defined here has to the best of our knowledge first been suggested by Lopez (1991) (see also Lopez et al. 1992) under the name *energetic reasoning*. Schwindt (1997) has independently developed a concept of interval work. He and, independently, Baptiste et al. (1998) were the first to answer Question 2. The area of the rectangles defined by activity processing time and resource requirements can be interpreted as work or energy, and we use the terms interchangeably.

Although our focus is primarily on the use of interval consistency tests for deducing domain reductions, we remark that Constraint 2 can, of course, also serve to derive lower bounds for makespan minimisation problems in the following way: Impose a hypothetical upper bound UB on the makespan; if this leads to a violation of Constraint 2 then $UB + 1$ is a lower bound. This approach is for example used by Nuijten (1994), Pesch and Tetzlaff (1996), Klein and Scholl (1997), Schwindt (1997), Heilmann and Schwindt (1997), and has been called destructive improvement due to the principle of refuting hypothetical constraints. Test values for UB are usually chosen through a dichotomising search. This type of bound argument can, equivalently, be applied whenever any of the tests described in the following sections causes an activity domain to become empty.

Finally, let us point out that resource capacity constraints in the form of Constraint 2, but mostly limited to intervals defined by earliest start and latest completion times of activities, have often been used in constraint logic based scheduling; see e.g. the description of solving a famous bridge scheduling problem (an RCPSP/max instance) in Van Hentenryck (1989) or the implementation of the cumulative constraint in CHIP (Aggoun and Beldiceanu 1993).

4 Consistency tests for disjunctive scheduling

The idea behind all tests described in the following subsections is to consider subsets $\mathcal{A} \subseteq \mathcal{V}_k$ of activities to be processed by the same resource k. Within these subsets, all possible activity sequences with a particular property are examined, e.g. the property that the sequence does not start with an activity $i \in \mathcal{A}$. If all such sequences are infeasible, then we can draw the conclusion that the sequence must *not* have this property and deduce that i must be first in $\mathcal{A}$.

The consistency tests are presented in order from strongest to weakest condition. While a stronger condition allows a stronger conclusion, it is at the same time more likely to be inapplicable. After developing the individual tests we generalise the results and show how they relate to the concept of interval consistency.

All tests in this section are derived for disjunctive scheduling problems where all activities $i \in \mathcal{A} \subseteq \mathcal{V}_k$ are mutually exclusive in the sense that they exclusively occupy resource k throughout their processing time. This is of course always the case if all activity sizes and resource capacities are equal to one, as in many machine scheduling problems, for example in the JSP. However, even if $r_{ik} > 1$ for some $i \in \mathcal{V}_k$, and consequently $R_k > 1$ the tests may still be used for subsets of "disjunctive" activities. For instance, it is likely that difficult RCPSP instances contain such disjunctive scheduling elements (Kolisch et al. 1995). Hence, disjunctive consistency tests have been suggested for the makespan minimisation RCPSP (Brucker et al. 1996b, Klein and Scholl 1997, Baptiste et al. 1998) and RCPSP/max (Schwindt 1997).

4.1 Input/output test

Figure 3 shows an example with a set $\mathcal{A} = \{i, j, k\}$ of three activities to be processed by the same resource. We can deduce that i must be scheduled first in the following way: Suppose i does not start first. Then all three operations must be processed in the interval $[2, 9)$. This means that a total processing time of $8 = 3 + 2 + 3$ must be scheduled in $7 = 9 - 2$ available time units, which is a contradiction. Thus we can conclude that i must start first; we can then deduce that start times of i greater than 1 can be removed from δ_i. Note that this conclusion cannot be drawn by separately considering any two of the three activities. Carlier and Pinson (1989) have formalised

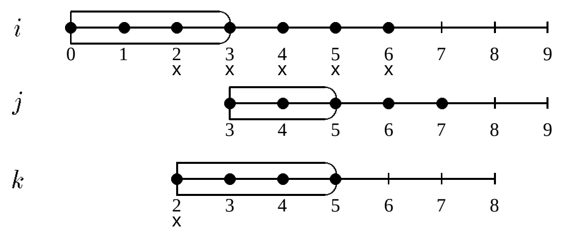

Figure 3 An example for the input test

the observation made in the example and have derived conditions under which it can be concluded that an operation $i \in \mathcal{A}$ must be scheduled first or last in $\mathcal{A}$. If i is scheduled before or after $\mathcal{A} \setminus \{i\}$ we may also think of i as the input or output of $\mathcal{A} \setminus \{i\}$, hence the name of the conditions. We use the shorthand notation $P(\mathcal{A}) = \sum_{i \in \mathcal{A}} p_i$ for the total processing time of $\mathcal{A}$.

Theorem 1 (Input/output). *Let $i \in \mathcal{A} \subseteq \mathcal{V}_k$. If*

$$\max_{u \in \mathcal{A} \setminus \{i\}, v \in \mathcal{A}, u \neq v} (lct_v - est_u) < P(\mathcal{A}) \tag{3}$$

then i must precede all activities in $\mathcal{A} \setminus \{i\}$ (input condition). Likewise, if

$$\max_{u \in \mathcal{A}, v \in \mathcal{A} \setminus \{i\}, u \neq v} (lct_v - est_u) < P(\mathcal{A}) \tag{4}$$

then i must succeed all operations in $\mathcal{A} \setminus \{i\}$ (output condition).

Proof. If i does not precede $\mathcal{A} \setminus \{i\}$, then all activities in $\mathcal{A}$ must be scheduled within $\max_{u \in \mathcal{A} \setminus \{i\}, v \in \mathcal{A}, u \neq v} (lct_v - est_u)$ time units. If Condition 3 holds this is not possible. The second part can be shown symmetrically. □

If the input or output condition holds we face the question how to adjust activity domains. Assume we have concluded that $\mathcal{A} \setminus \{i\} \rightarrow i$. Clearly, i can only start after the minimum completion time t^* of all activities in $\mathcal{A} \setminus \{i\}$. Unfortunately, finding t^*

is an $\mathcal{NP}$-hard problem: It is equivalent to solving the one-machine makespan minimisation problem with release time and due dates (Carlier 1982). Therefore we resort to approximating t^*. A simple and obvious approximation is the maximal earliest completion time in $\mathcal{A} \setminus \{i\}$. However, we can do better by considering the preemptive relaxation of the one-machine problem (preemptive bound). For this problem, an optimal solution known as Jackson's Preemptive Schedule (JPS) can be efficiently obtained by scheduling the activities in $\mathcal{A} \setminus \{i\}$ "from left to right" according to the "earliest due date" priority dispatching rule (Jackson 1956): Whenever the resource is free, schedule the activity i with minimal lct_i; if an activity j with $lct_j < lct_i$ becomes available while i is in process then interrupt i and start j. We denote the completion time of JPS for $\mathcal{A} \setminus \{i\}$ by $Ect(\mathcal{A} \setminus \{i\})$. Clearly, $Ect(\mathcal{A} \setminus \{i\})$ is a lower bound on the earliest start of i, and the same holds true for all subsets $\mathcal{A}' \subseteq \mathcal{A} \setminus \{i\}$. However, Carlier (1982) has shown that

$$Ect(\mathcal{A} \setminus \{i\}) = \max_{\mathcal{A}' \subseteq \mathcal{A} \setminus \{i\}} \{\min_{u \in \mathcal{A}'} est_u + P(\mathcal{A}')\}. \tag{5}$$

This implies that $Ect(\mathcal{A}') \leq Ect(\mathcal{A} \setminus \{i\})$, if $\mathcal{A}' \subset \mathcal{A} \setminus \{i\}$. We can now state a domain reduction rule to be applied if the output condition holds:

$$\delta_i := \delta_i \setminus \{t | t < Ect(\mathcal{A} \setminus \{i\})\} \tag{6}$$

A symmetric rule applies if the input condition holds. Using $Lst(\mathcal{A} \setminus \{i\})$ as the preemptive bound for the latest start time of $\mathcal{A} \setminus \{i\}$, obtained by preemptively scheduling the activities in $\mathcal{A} \setminus \{i\}$ "from right to left" as late as possible according to the "maximum latest start" priority dispatching rule, the domain reduction rule is:

$$\delta_i := \delta_i \setminus \{t | t > Lst(\mathcal{A} \setminus \{i\}) - p_i\} \tag{7}$$

In addition to these domain reductions we can adjust lct_j to be less than or equal to lst_i (output) or est_j to be greater than or equal to ect_i (input) for all $j \in \mathcal{A} \setminus \{i\}$. Alternatively, we can wait until a subsequent application of this test for a set $\mathcal{A} = \{i, j\}$ automatically yields this reduction.

Before returning to the initial example, let us point out that — like all consistency tests — the input/output test consists of (a) conditions and (b) corresponding domain reduction rules.

For the example in Figure 3 the maximum of the expression on the left side of the input condition is $9 - 2$, and $P(\mathcal{A}) = 8$; since $9 - 2 < 8$, we can deduce $i \rightarrow \{j, k\}$. With $Lst(\{j, k\}) = 4$ the domain of i becomes $\delta_i := [0, 6] \setminus \{t | t > 4 - 3\} = [0, 1]$. Note the effect of using the preemptive bound: By using $Lst(\{j, k\})$ we have obtained a stronger domain reduction for i than we would have by considering lst_j and lst_k separately, which would have left the value 2 in δ_i. Finally, the domain of k becomes $\delta_k := [2, 5] \setminus \{t | t < ect_i\} = [3, 5]$. As pointed out above, this reduction in δ_k could also be achieved through a further application of the input test for $\mathcal{A} = \{i, k\}$.

The input condition also applies in the example in Figure 1. For activity i and $\mathcal{A} = \{i, j\}$, we obtain $8 - 2 < 7$ and deduce $i \rightarrow j$. The domain of i remains unmodified, and the domain of j reduces to $\delta_j := \delta_j \setminus \{t | t < 3\} = [3, 8]$.

In branch and bound procedures that branch over disjunctive edges, the test may be employed to immediately select the orientation of edges, a process often called immediate selection, as first suggested by Carlier and Pinson (1989), or edge finding, a term introduced by Applegate and Cook (1991). The input/output test was first described by Carlier and Pinson in the context of a branch and bound algorithm for the JSP; the tests that they actually implemented in their initial algorithm were limited to two-element sets $\mathcal{A}$ and one additional heuristically determined $\mathcal{A}$ and $i \in \mathcal{A}$ for each resource. Using these tests, they were for the first time able to optimally solve a notoriously difficult 10×10 JSP instance (Fisher and Thompson 1963) that — despite many attempts — had defied solution for over 25 years.

Efficient algorithms that have later been developed for testing the input/ output conditions for all $\mathcal{A}$ and i and performing the corresponding domain reductions based on the preemptive bounds usually use an ordering of activities according to earliest start and latest completion times. The challenging part is to test the input/output conditions and calculate preemptive bounds at the same time. Carlier and Pinson (1990), Martin and Shmoys (1996), and Nuijten (1994) have designed $O(|\mathcal{V}_k|^2)$ algorithms for testing all subsets $\mathcal{A} \subseteq \mathcal{V}_k$. The algorithm of Nuijten has the interesting property that it can be generalised for cumulative scheduling. $O(|\mathcal{V}_k| \log |\mathcal{V}_k|)$ algorithms for testing all subsets have been described by Brucker et al. (1996a) and Carlier and Pinson (1994). Caseau and Laburthe (1994, 1995, 1996) describe an algorithm based on the concept of "task intervals" for checking all sets $\mathcal{A}$ with effort $O(|\mathcal{V}_k|^3)$. The advantage of their approach is that the consistency conditions can be evaluated incrementally within a search procedure. When used within a branch-and-bound algorithm this means that the effective time complexity for performing the tests at each node of the search tree is usually lower than $O(|\mathcal{V}_k|^3)$ because it is not necessary to test all $\mathcal{A}$; although the worst case complexity for performing the tests at a node is still $O(|\mathcal{V}_k|^3)$, the average complexity is lower. This contrasts with the usual approach of applying the full test at each node of a branch-and-bound tree.

Finally, we would like to mention that to our knowledge all algorithms discussed above do not test the input/output conditions in the form of Theorem 1, where we have required in the maximum expressions that $u \neq v$, but rather allow for $u = v$, thus actually testing a weaker condition. Although the extension may seem trivial it does lead to additional deductions in certain cases. However, it is not always obvious how to include it in existing algorithms without increasing their time complexity.

4.2 Input-or-output test

From the input/output condition it can be deduced that an operation $i \in \mathcal{A} \subseteq \mathcal{V}_k$ must be scheduled first or last in $\mathcal{A}$. The weaker input-or-output condition can be used to show that a precedence relation $i \rightarrow j$ must exist between a pair of activities i and j from set $\mathcal{A}$.

Figure 4 shows an example with a set $\mathcal{A} = \{i, j, k, l\}$ of four activities to be processed by the same resource. The input/output condition does not allow to draw any conclusions about the order in which the activities must be scheduled. However, we can deduce that i must precede j: Suppose i is not scheduled first *and* j is not scheduled last. Then all four activities with a total processing time of $7 = 3 + 2 + 1 + 1$

must be scheduled within the interval $[2, 8]$, which is a contradiction. Hence we can conclude that it is impossible that at the same time i is not first *and* j is not last. If either i must be first or j must be last, then i must precede j, and we can remove the start time 3 from δ_j. This observation leads to the following theorem.

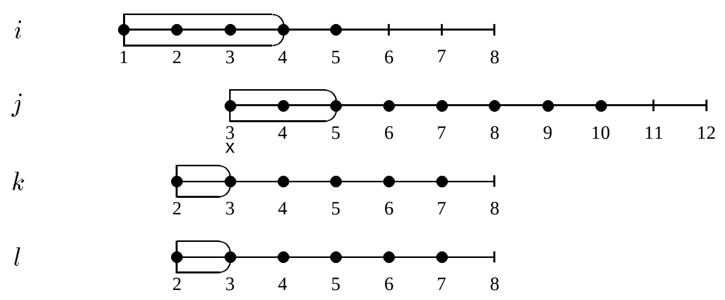

Figure 4 An example for the input-or-output test

Theorem 2 (Input-or-output). *Let $i, j \in \mathcal{A} \subseteq \mathcal{V}_k$. If*

$$\max_{u \in \mathcal{A}\setminus\{i\}, v \in \mathcal{A}\setminus\{j\}, u \neq v} (lct_v - est_u) < P(\mathcal{A}) \tag{8}$$

then i must be scheduled first or j must be scheduled last in $\mathcal{A}$. If $i \neq j$ then i must precede j.

Proof. Suppose neither i is scheduled first nor j is scheduled last. All activities in $\mathcal{A}$ must then be scheduled within $\max_{u \in \mathcal{A}\setminus\{i\}, v \in \mathcal{A}\setminus\{j\}, u \neq v}(lct_v - est_u)$ time units. If Condition 8 holds, this is impossible and we can conclude that either i must be first or j must be last in $\mathcal{A}$. In both cases i must precede j if $i \neq j$. □

Comparison to the very similar input/output conditions shows in what sense the input-or-output condition is weaker. If this condition holds and $i \neq j$ which means that $i \rightarrow j$, we can reduce the domains of i and j in the following way:

$$\delta_i := \delta_i \setminus \{t | t > lst_j - p_i\} \tag{9}$$

$$\delta_j := \delta_j \setminus \{t | t < ect_i\} \tag{10}$$

If the condition holds for $i = j$, the new domain of i after reduction is:

$$\delta_i := \delta_i \setminus \{t | Lst(\mathcal{A} \setminus \{i\}) - p_i < t < Ect(\mathcal{A} \setminus \{i\})\} \tag{11}$$

While the domain reduction rules 9 and 10 always lead to a domain reduction at the domain bounds if any, Rule 11 may remove values within the domain but leaves the bounds untouched.

For the example in Figure 4 we obtain $8 - 2 < 7$ and deduce $i \rightarrow j$. By applying domain reduction rule 10 we can remove the value 3 from δ_j. Figure 5 shows another example where the input-or-output condition can deduce that a single activity must either start first or last; in terms of Theorem 2 this is the case where $i = j$. We obtain $5 - 3 < 4$ and conclude that i must start before or after $\{k, l\}$. Domain reduction rule 11 allows to remove the values $[2, 4]$ from δ_i. As a final example, note that the

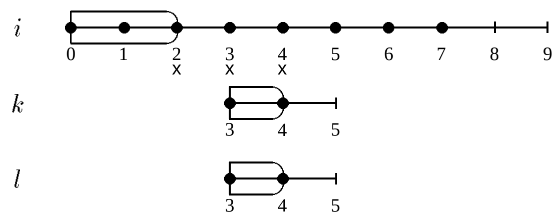

Figure 5 Input-or-output condition example: i must be first or last

result $i \rightarrow \{j, k\}$ that we have obtained with the input/output condition for the example in Figure 3 can also be deduced in two steps with the input-or-output condition, resulting in the conclusions $i \rightarrow j$ and $i \rightarrow k$. However, the corresponding reduction in δ_i is weaker, leaving the value 2 in δ_i.

To our knowledge, the input-or-output test in its general form has not been discussed in the literature. A similar condition for the special case where $i = j$ has been described by Carlier and Pinson (1990); see also Błażewicz et al. (1997, 1998). Stronger conditions based on considering all sets $\mathcal{A}$ of cardinality r, hence called r-set conditions, have been discussed by Brucker et al. (1996a). They describe an $O(|\mathcal{V}_k|^2)$ 3-set algorithm that checks all activity sets of cardinality three and detects all pairwise ordering relations derivable from triples. The algorithm thus implements the input-or-output test for $|\mathcal{A}| = 3$. Judging from the implementation within their branch-and-bound procedure for the JSP, the efficiency of the 3-set tests is comparable to that of the input/output tests. It is unclear whether a low polynomial time-complexity r-set algorithm could be developed for $r > 3$.

The development of an algorithm with low polynomial time complexity for testing the input-or-output conditions is an open issue. Based on experience with other consistency tests, we conjecture that in order to be of practical value such an algorithm must at most have time complexity $O(|\mathcal{V}_k|^2)$. There is an obvious $O(|\mathcal{V}_k|^4)$ algorithm using task or activity intervals, and we have designed an $O(|\mathcal{V}_k|^3)$ algorithm.

4.3 Input/output negation test

By further relaxing the condition to be tested, we can still draw additional conclusions in situations where the input-or-output condition and the stronger input/output conditions do not hold. Figure 6 shows an example with a set $\mathcal{A} = \{i, j, k\}$ of three activities to be processed by the same resource. Although we cannot conclude that activity i must be last or must precede j or k, we can deduce that i must *not be first,* and therefore remove the value 2 from δ_i. By generalising the observations made in

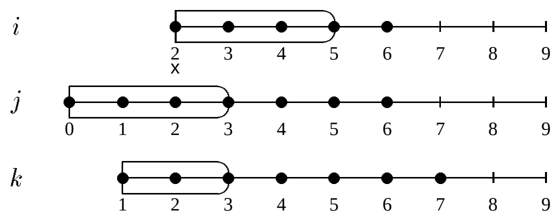

Figure 6 An example for the input negation test

the example, we arrive at the following theorem.

Theorem 3 (Input/output negation). *Let $i \in \mathcal{A} \subseteq \mathcal{V}_k$. If*

$$\max_{v \in \mathcal{A} \setminus \{i\}} (lct_v - est_i) < P(\mathcal{A}) \tag{12}$$

then i must not start first in $\mathcal{A} \setminus \{i\}$ (input negation). If

$$\max_{u \in \mathcal{A} \setminus \{i\}} (lct_i - est_u) < P(\mathcal{A}) \tag{13}$$

then i must not end last in $\mathcal{A} \setminus \{i\}$ (output negation).

Proof. If i precedes $\mathcal{A} \setminus \{i\}$, all activities in $\mathcal{A}$ must be scheduled in the interval $[est_i, \max_{v \in \mathcal{A} \setminus \{i\}} lct_v)$. If Condition 12 holds, this is not possible. The second part can be shown symmetrically. □

Again, it is easy to see in which sense these conditions are weaker than in the preceding tests. Domain reduction rules can be based on the observation that i must succeed (input negation) or precede (output negation) at least one other activity in $\mathcal{A}$:

$$\delta_i := \delta_i \setminus \{t | t < \min_{u \in \mathcal{A} \setminus \{i\}} ect_u\} \tag{14}$$

$$\delta_i := \delta_i \setminus \{t | t > \max_{u \in \mathcal{A} \setminus \{i\}} lst_u - p_i\} \tag{15}$$

For the example in Figure 6 the input negation condition yields $9 - 2 < 8$ and we conclude $i \not\to \{j, k\}$. According to domain reduction rule 14 we can therefore remove all values less than 3 from δ_i.

Conclusions similar to those obtained in the examples for the input/output and input-or-output test could also have been produced through successive application of the input/output negation test. Since the condition to be tested for the input/output negation conclusion is weaker than the preceding conditions, it will of course hold whenever the stronger conditions apply. Consider again the example in Figure 3. Here, the input/output negation conditions allows to conclude $j \not\rightarrow \{i, k\} \Leftrightarrow i \rightarrow j \vee k \rightarrow j$ and $k \not\rightarrow \{i, j\} \Leftrightarrow i \rightarrow k \vee j \rightarrow k$, which implies $i \rightarrow \{j, k\}$. However, this implication is not automatically deduced by the input/output negation condition. This demonstrates that input/output negation conditions alone do not deduce all interesting domain reductions. A similar effect can be seen in the example in Figure 4. Here, the input/output negation conditions can be used to deduce $\{j, k, l\} \not\rightarrow i$ and $j \not\rightarrow \{i, k, l\}$, but this does not allow to remove the value 3 from δ_j as in the input-or-output test.

The input/output negation test has first been suggested by Carlier and Pinson (1989). Most authors working on consistency tests have considered the test in some form. However, an algorithm that tests all interesting $\mathcal{A}$ and i with effort $O(|\mathcal{V}_k|^2)$ has only recently been developed by Baptiste and Le Pape (1996). Other researchers have often applied the tests in an incomplete way, testing only some $\mathcal{A}$ and i (Carlier and Pinson 1989, 1990, Nuijten 1994, Baptiste and Le Pape 1995). Caseau and Laburthe (1994, 1995) have integrated the tests in their task interval algorithm which tests input/output conditions and the negation conditions with effort $O(|\mathcal{V}_k|^3)$.

4.4 Summary and generalisation

All disjunctive interval consistency conditions that we have discussed can be regarded as special cases of the following theorem.

Theorem 4 (Sequence consistency). *Let $\mathcal{A}', \mathcal{A}'' \subset \mathcal{A} \subseteq \mathcal{V}_k$. If*

$$\max_{u \in \mathcal{A} \setminus \mathcal{A}', v \in \mathcal{A} \setminus \mathcal{A}'', u \neq v} (lct_v - est_u) < P(\mathcal{A}) \tag{16}$$

then an activity in $\mathcal{A}'$ must start first or an activity in $\mathcal{A}''$ must end last in $\mathcal{A}$.

Proof. If none of the activities in $\mathcal{A}''$ succeeds $\mathcal{A} \setminus \mathcal{A}''$ *and* none of the activities in $\mathcal{A}'$ precedes $\mathcal{A} \setminus \mathcal{A}'$, then $\mathcal{A}$ must be processed within $\max_{u \in \mathcal{A} \setminus \mathcal{A}', v \in \mathcal{A} \setminus \mathcal{A}'', u \neq v} (lct_v - est_u)$ units of time. If Condition 16 holds this is a contradiction. $\square$

The results of the preceding sections are summarised in Table 2. For each condition, the table shows the values of $\mathcal{A} \setminus \mathcal{A}'$ and $\mathcal{A} \setminus \mathcal{A}''$ that, when used in Theorem 4, yield the condition. The conclusions of Theorem 4 have been reformulated to match the tests presented above. Note that the conclusion is always the negation of the hypothesis $\mathcal{H}$ falsified by the test.

4.5 Relation to general interval consistency

We will now relate the Sequence Consistency Theorem to the general interval consistency constraint 2. For disjunctive scheduling, Constraint 2 reduces to $t_2 - t_1 \geq P(\mathcal{V}_k, t_1, t_2)$. The following theorem shows how we can efficiently test violations of this constraint.

Test	$\mathcal{A} \setminus \mathcal{A}''$	$\mathcal{A} \setminus \mathcal{A}'$	*Conclusion* ($\neg\mathcal{H}$)
input	$\mathcal{A}$	$\mathcal{A} \setminus \{i\}$	$i \rightarrow \mathcal{A} \setminus \{i\}$
output	$\mathcal{A} \setminus \{i\}$	$\mathcal{A}$	$\mathcal{A} \setminus \{i\} \rightarrow i$
input-or-output	$\mathcal{A} \setminus \{j\}$	$\mathcal{A} \setminus \{i\}$	$i \rightarrow \mathcal{A} \setminus \{i\} \vee \mathcal{A} \setminus \{j\} \rightarrow j$
input negation	$\mathcal{A} \setminus \{i\}$	$\{i\}$	$i \not\rightarrow \mathcal{A} \setminus \{i\}$
output negation	$\{i\}$	$\mathcal{A} \setminus \{i\}$	$\mathcal{A} \setminus \{i\} \not\rightarrow i$

Table 2 Summary of disjunctive interval consistency tests, $\mathcal{A}', \mathcal{A}'' \subset \mathcal{A} \subseteq \mathcal{V}_k$

Theorem 5. *If, for some time interval* $[t_1, t_2)$,

$$t_2 - t_1 < P(\mathcal{V}_k, t_1, t_2), \tag{17}$$

then

$$\max_{i,j \in \mathcal{V}_k(t_1,t_2), i \neq j} (lct_j - est_i) < P(\mathcal{V}_k(t_1, t_2)) \tag{18}$$

Proof. From Equation 1 we know that $0 < t_2 - t_1 < P(\mathcal{V}_k, t_1, t_2)$ implies that $|\mathcal{V}_k(t_1, t_2)| \geq 2$. We consider two activities $i, j \in \mathcal{V}_k(t_1, t_2), i \neq j$, and start to transform Condition 17 into Condition 18 by rewriting the left hand side of 17:

$$lst_j + t_2 - lst_j - t_1 < P(\mathcal{V}_k, t_1, t_2),$$

By observing that $t_2 - lst_j \geq p_j(t_1, t_2) \geq 0$, according to Equation 1, we can approximate the left side. We rewrite the right side and obtain:

$$lst_j + p_j(t_1, t_2) - t_1 < p_i(t_1, t_2) + p_j(t_1, t_2) + \sum_{l \in \mathcal{V}_k \setminus \{i,j\}} p_l(t_1, t_2)$$

Again, we know from Equation 1 that $ect_i - t_1 \geq p_i(t_1, t_2) \geq 0$. This approximation leads to:

$$lst_j - t_1 < ect_i - t_1 + \sum_{l \in \mathcal{V}_k \setminus \{i,j\}} p_l(t_1, t_2)$$

Next, we approximate the sum on the right hand side, once again using Equation 1 which tells us that $p_k \geq p_k(t_1, t_2) \geq 0$, and obtain:

$$lst_j - ect_i < \sum_{l \in \mathcal{V}_k(t_1,t_2) \setminus \{i,j\}} p_l$$

By adding $p_i + p_j$ on both sides we arrive at:

$$lct_j - est_i < P(\mathcal{V}_k(t_1, t_2))$$

As it is always possible to choose i and j in such a way that the maximum difference $lct_j - est_i$ is realised, Condition 18 must hold. $\square$

The theorem tells us two interesting things. First, it states that if an interval capacity constraint is violated for some arbitrary time interval $[t_1, t_2)$, then there will also be a violation for an interval defined by the earliest start and latest completion time of two activities in $\mathcal{V}_k(t_1, t_2)$. When checking for violations this allows us to restrict our attention to intervals defined by earliest start and latest completion times — called task or activity intervals (Caseau and Laburthe 1994) — instead of considering all possible time intervals. Inconsistencies can thus be detected by testing $O(|\mathcal{V}_k|^2)$ intervals. This answers the initial question (posed in Section 3) what intervals we should test. Second, the theorem states that, as long as we test all activity intervals, there is nothing to be gained from considering *interval* processing time instead of *simple* processing time. If interval processing time has an effect on the test for a given set $\mathcal{A}$ then we can obtain the same effect by considering a different set $\mathcal{A}'$. In summary, this means that an algorithm which tests Condition 18 for all activity intervals will detect all violations according to the more general concept of Condition 17 which is the negation of the disjunctive version of the general interval consistency constraint 2.

It is worth emphasising that this statement is independent of the particular set of hypothetical constraints $\mathcal{H}$ to be tested. This can be seen as follows: For any set of constraints, it is always possible to first add and propagate the constraints, and then test the interval consistency constraints. The particular form of the sequence consistency tests is simply an accelerated version of this “add and propagate, then test” process. For illustration, consider again the example shown in Figure 3, where the conclusion $i \rightarrow \{j, k\}$ could also have been obtained in the following way: (1) Add $\mathcal{H} = \{i \not\rightarrow \{j, k\}\}$, (2) update the domain of i based on $\mathcal{H}$, which yields $\delta_i := \delta_i \backslash \{t | t < \min_{u \in \{j,k\}} ect_u\} = [5, 6]$, and (3) test the interval consistency constraint 2 for the activity interval defined by $\{i, j, k\}$ which has the left time bound 2 and the right time bound 9. Because $9 - 2 \not\geq 8$ this test fails and we conclude $\neg\mathcal{H} \Leftrightarrow i \rightarrow \{j, k\}$.

For disjunctive scheduling, Theorem 5 improves the results obtained by Schwindt (1997) and Baptiste et al. (1998) for the cumulative case discussed in Section 5.

4.6 2-consistency and 2-bound-consistency

This section relates the consistency tests to the general concept of 2-consistency and 2-bound-consistency that is commonly used in the constraint satisfaction and constraint propagation literature (Kumar 1992, Tsang 1993, Nuijten 1994, Van Hentenryck 1989). We derive a 2-consistency test and show that the sequencing tests can be used to achieve 2-bound-consistency.

We use the following informal consistency definitions: Activity domains are called 2-consistent if, for any pair $i, j \in \mathcal{V}$, and for any value $x \in \delta_i$ there is some value $y \in \delta_j$ such that $st_i = x$ and $st_j = y$ is permitted by the constraints of the scheduling problem. The weaker definition of bound consistency looks at domain bounds: Activity domains are called 2-bound-consistent, or 2-b-consistent for short, if, for any pair $i, j \in \mathcal{V}$, and for every value $x \in \{\min \delta_i, \max \delta_i\}$ there is a value $y \in \delta_j$ such that $st_i = x$ and $st_j = y$ is permitted. Clearly, 2-consistency implies 2-b-consistency.

The concept of bound consistency is of interest because, as we have seen, many consistency tests are based on domain bound considerations. In addition, the propagation of temporal constraints depends on domain bounds. Any change in domain bounds is therefore likely to trigger further domain reductions. Finally, if domains are approximated by start time windows — and this is often done for reasons of implementation efficiency — bound-consistency is the only reasonable concept of consistency.

Figure 7 shows an example, taken from Nuijten (1994), with a pair of activities $i, j \in \mathcal{V}_k$ where any 2-inconsistent value is marked. For example, j cannot start at

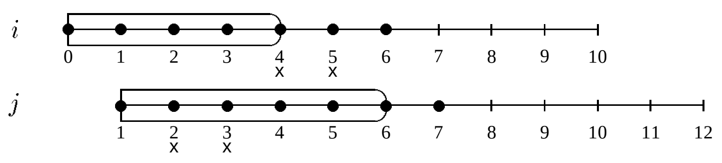

Figure 7 2-consistency

time 2 since this does neither leave enough room for i to be processed before j nor after j. In general, i cannot start in the open interval $(lst_j - p_i, ect_j)$. Note that the interval can be empty if $ect_j \leq lst_j - p_i$. The observation is summarised in the following theorem due to Nuijten (1994).

Theorem 6 (2-consistency). *Let $i, j \in \mathcal{V}_k, i \neq j$. δ_i and δ_j are 2-consistent if and only if*

$$\delta_i \cap (lst_j - p_i, ect_j) = \emptyset. \tag{19}$$

Proof. If j is started at time $t \in \delta_j$ then i is blocked during the open interval $(t - p_i, t + p_j)$. The left bound of the interval is maximal for $t = lst_j$, and the right bound is minimal for $t = est_j$. Thus the minimal interval during which i cannot start is $(lst_j - p_i, ect_j)$. All other possible start times of j leave possible start times for i. □

An $O(|\mathcal{V}_k|^2)$ algorithm for achieving 2-consistency is described by Nuijten (1994).

The following result shows that the sequence consistency tests based on Theorem 4 can be used to ensure 2-b-consistency.

Theorem 7. *The input/output, input-or-output, and input/output negation tests all achieve 2-b-consistency.*

Proof. For $\mathcal{A} = \{i, j\}$ all the tests simplify to:

$$lct_j - est_i < p_i + p_j \Rightarrow j \rightarrow i \Rightarrow \begin{cases} \delta_i & := & \delta_i \setminus \{t | t < ect_j\} \\ \delta_j & := & \delta_j \setminus \{t | t > lst_i - p_j\} \end{cases}$$

To achieve 2-b-consistency any 2-inconsistent value must be removed from the domain bounds. According to Equation 19, the left domain bound can only be 2-inconsistent if

$$lst_j - p_i < est_i \Leftrightarrow lct_j - est_i < p_i + p_j.$$

In this case, the condition of the tests is satisfied and any inconsistent values are removed by the first domain reduction rule above. The proof for the right domain bound is symmetrical. □

4.7 Sequence consistency does not imply k-consistency

Trivial though it may be, it is worth emphasising that the consistency tests only check necessary, but not sufficient conditions for the existence of a feasible schedule and that the domain reduction rules employed are heuristic. While we could show that the sequence consistency tests always achieve 2-b-consistency, this means that they in general do not achieve k-b-consistency for $k > 2$. The example in Figure 8 illustrates these two points.

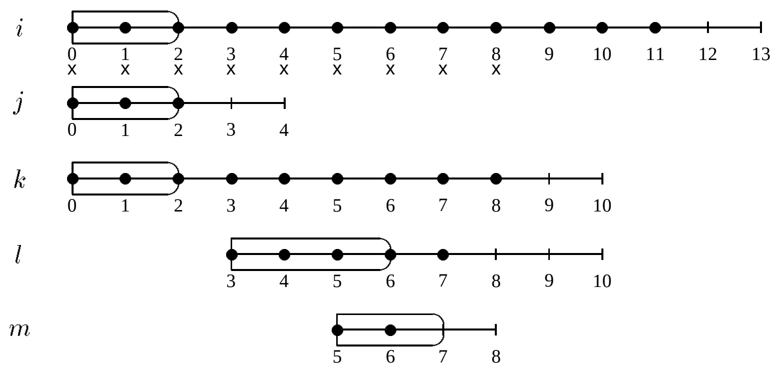

Figure 8 When sequence consistency tests fail

In the example, $\mathcal{A} = \{i, j, k, l, m\}$. The output condition allows to conclude $\{j, k, l, m\} \rightarrow i$, since $10 - 0 < 11$. The preemptive bound $Ect(\mathcal{A} \setminus \{i\})$ for the earliest completion time of $\{j, k, l, m\}$ is 9. According to domain reduction rule 6 this leaves the value 9 as the left bound of δ_i. However, manual inspection shows that the earliest completion time of $\{j, k, l, m\}$ is actually 10. Thus, the input/output test leaves an inconsistent value at the left bound of δ_i. This demonstrates that the domain reduction rule based on the preemptive bound is heuristic.

Now modify the example by reducing lct_i to 11. The input/output test still yields the same result, and none of the other sequence consistency tests leads to an inconsistency (by producing an empty domain). Again, manual inspection shows that there is no feasible schedule for $\mathcal{A}$.

4.8 Shaving

In the tests based on the Sequence Consistency Theorem 4 we have tried to refute hypothetical constraints on the sequence in which activities in a set $\mathcal{A} \subseteq \mathcal{V}_k$ execute. Now, we take a purely time-oriented approach and consider hypothetical constraints on individual activity start times. If we can falsify such a constraint, then we can reduce the corresponding activity domain in an obvious way. The process of reducing activity domains based on this kind of reasoning has been called *shaving* (Martin and Shmoys 1996, Caseau and Laburthe 1996).

For example, we can test a hypothetical constraint of the type $st_i > t_x$ for some $t_x \in \delta_i$. If this leads to a contradiction, then we can conclude that st_i must be less than or equal to t_x and remove all values greater t_x from δ_i. A contradiction may be caused by a direct violation of the interval capacity constraint 2 or after propagating the hypothetical constraint by repeatedly applying other consistency tests. Values of t_x can for example be chosen by a dichotomising search over δ_i.

A shaving approach for disjunctive scheduling has been proposed by Carlier and Pinson (1994) for solving the JSP. Martin and Shmoys (1996) have, independently, applied the technique within a time-oriented branch-and-bound algorithm for the JSP. Using a shaving technique, Caseau and Laburthe (1996) were able to obtain a proof of optimality for the famous 10×10 JSP due to Fisher and Thompson (1963) with only 7 backtracks.

5 Consistency tests for cumulative scheduling

While disjunctive scheduling or sequencing is concerned with unit resource requirements and capacities, cumulative scheduling considers the general case of arbitrary requirements and capacities.

5.1 Activity interval consistency

The disjunctive sequence consistency tests developed in the preceding section can be generalised for cumulative scheduling in a straightforward way by considering interval work instead of processing times. This relation was first pointed out by Nuijten (1994) (see also Nuijten and Aarts 1996). The following rule extends Theorem 4 for cumulative scheduling. As the time intervals considered are activity or task intervals defined by activity sets, we have chosen the name activity interval consistency.

Theorem 8 (Activity interval consistency). *Let $\mathcal{A}', \mathcal{A}'' \subset \mathcal{A} \subseteq \mathcal{V}_k$.If*

$$R_k \cdot \max_{u \in \mathcal{A} \setminus \mathcal{A}', v \in \mathcal{A} \setminus \mathcal{A}''} (lct_v - est_u) < W(\mathcal{A}) \tag{20}$$

then an activity in $\mathcal{A}'$ must start first or an activity in $\mathcal{A}''$ must end last.

Proof. Similar to proof of Theorem 4. □

In contrast to sequencing, we can no longer assume that $i \neq j$ because an activity that starts first may now also end last. Comparison of Condition 20 to the general interval capacity constraint 2 shows that the condition only considers activity or task intervals and has a weaker right hand side. In the disjunctive case we were able to show that it was sufficient to consider activity intervals and that there was nothing to be gained from using interval work instead of set based work on the right side. However, it turns out that this is not the case for cumulative scheduling, so that the condition can actually be strengthened. The reason for presenting the condition in the above form is that this extension of the disjunctive case allows to generalise algorithms originally designed for sequencing. We will discuss a sharper form in Section 5.2.

The theorem can be used to derive tests in analogy to the sequencing tests by using suitable values for $\mathcal{A}'$ and $\mathcal{A}''$, as shown in Table 2. Note that the meaning of conclusions such as $\mathcal{A} \setminus \{i\} \rightarrow i$ or $i \rightarrow \mathcal{A} \setminus \{i\}$ is that i must end after (start before) all activities in $\mathcal{A} \setminus \{i\}$; in contrast to the disjunctive case this, however, does not imply that it must also start after (end before) $\mathcal{A} \setminus \{i\}$.

Useful domain reductions can be deduced for the cumulative version of the input-or-output test with $i = j$. For $\mathcal{A}' = \mathcal{A}'' = \{i\}$, i.e. for testing the hypothetical constraints $\mathcal{H} = \{\, i \not\rightarrow \mathcal{A} \setminus \{i\}, \mathcal{A} \setminus \{i\} \not\rightarrow i\}$, Theorem 8 yields:

$$R_k \cdot \max_{u \in \mathcal{A} \setminus \{i\}, v \in \mathcal{A} \setminus \{i\}} (lct_v - est_u) < W(\mathcal{A}) \Rightarrow i \rightarrow \mathcal{A} \setminus \{i\} \vee \mathcal{A} \setminus \{i\} \rightarrow i \tag{21}$$

Clearly, the excess amount of work that cannot be processed in the interval defined by $\min_{u \in \mathcal{A} \setminus \{i\}} est_u$ and $\max_{v \in \mathcal{A} \setminus \{i\}} lct_v$ is the difference of the total work required by $\mathcal{A}$ and the capacity available within the interval. Since only activity i can move partially or completely out of the interval, we can conclude that the amount of processing time of i to be moved outside to the left and/or right, denoted by rest$(\mathcal{A}, i)$, is:

$$\text{rest}(\mathcal{A}, i) = \lceil (W(\mathcal{A}) - R_k \cdot \max_{u \in \mathcal{A} \setminus \{i\}, v \in \mathcal{A} \setminus \{i\}} (lct_v - est_u))/r_{ik} \rceil. \tag{22}$$

This observation allows to deduce domain reductions if the minimum amount of processing time that is always outside of the interval, regardless of the chosen start time, is less than the required amount:

$$p_i - \max_{u \in \mathcal{A} \setminus \{i\}, v \in \mathcal{A} \setminus \{i\}} (lct_v - est_u) < \text{rest}(\mathcal{A}, i). \tag{23}$$

If Condition 23 holds, then the part of i that must be outside of the interval must either be completely on the left or be completely on the right side of the interval. This leads to the following domain reduction rule that can be applied if Conditions 21 and 23 hold:

$$\delta_i := \delta_i \setminus \{t | \min_{u \in \mathcal{A} \setminus \{i\}} est_u - \text{rest}(\mathcal{A}, i) < t < \max_{v \in \mathcal{A} \setminus \{i\}} lct_v + \text{rest}(\mathcal{A}, i) - p_i\} \tag{24}$$

This rule can actually be sharpened as follows: If the left or right bound reduction may be applied for $\mathcal{A} \setminus \{i\}$ then it can also be applied for all subsets $\mathcal{A}' \subseteq \mathcal{A} \setminus \{i\}$; this is not shown here. Note that the sharpened form of the rule is equivalent to domain reduction rule 11. We refer to the conditions and this domain reduction rule as the cumulative input-or-output test.

Figure 9 illustrates the test. It shows an example, taken from Nuijten (1994), with four activities to be processed by the same resource k with capacity $R_k = 2$.

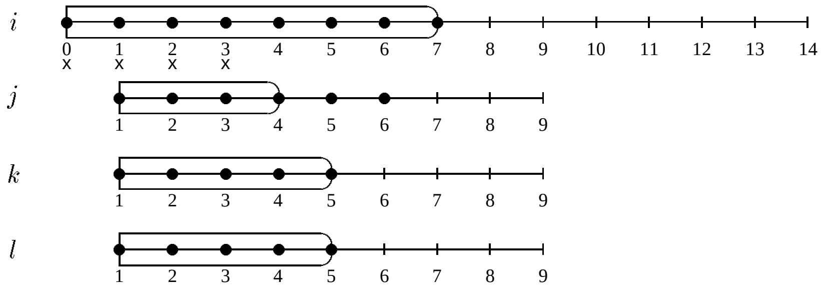

Figure 9 Four activities requiring 1 unit of a resource with capacity 2

Inspection shows that if activity i is started before time 4, then it is impossible to schedule all of the other activities j, k, l within their time window. This is detected by the input-or-output test in the following way: Because $2 \cdot (9-1) < 18$ we conclude that $i \rightarrow \{j,k,l\} \vee \{j,k,l\} \rightarrow i$. The amount of processing time of i that must take place outside of the interval $[1,9)$ is rest$(\{i,j,k,l\}, i) = \lceil (18 - 2 \cdot (9-1))/1 \rceil = 2$. Because $7 - 8 < 2$, Condition 23 is satisfied and we apply the domain reduction rule $\delta_i := \delta_i \setminus \{t | 1 - 2 < t < 9 + 2 - 7\} = [4,7]$, as shown in Figure 9.

It is interesting to consider a slight modification of the example: For $p_i = 6$ the reduction rule yields $\delta_i := \delta_i \setminus \{t | 0 < t < 4\} = \{0, 4, \ldots, 7\}$; the value 0 is thus left in δ_i and the domain bounds remain untouched.

The test presented here is similar to the three cumulative tests described by Nuijten (1994). Nuijten uses the tests to extend the disjunctive sequence consistency algorithm for the cumulative case. The time complexity of the resulting algorithm is $O(|\{r_{ik}\}| \cdot |\mathcal{V}_k|^3)$, where $|\{r_{ik}\}|$ is the number of distinct resource capacity requirements. As mentioned before, the tests could be strengthened by using interval work instead of simple work and by considering additional time intervals other than activity intervals. This is explained in the following section.

5.2 General interval consistency (energetic reasoning)

Figure 10 shows an example, similar to an example used by Baptiste et al. (1998), with five activities that require one unit of a resource with capacity 2. We can conclude that activity i must start after time 6. This can be deduced by testing the hypothetical constraint $\mathcal{H} = \{lct_i \leq 10\}$ against the general interval consistency constraint 2 for the interval $[1,9)$. If i is constrained to finish at time 10 or before, then the total amount of interval work to be processed within $[1,9)$ is $2 \cdot 4 + 3 \cdot 3 = 17$ units, whereas only $2 \cdot (9-1) = 16$ units are available. We can thus conclude $\neg\mathcal{H}$ and remove values less than or equal to 6 from δ_i. We emphasise that $\mathcal{H}$ can only be refuted by testing

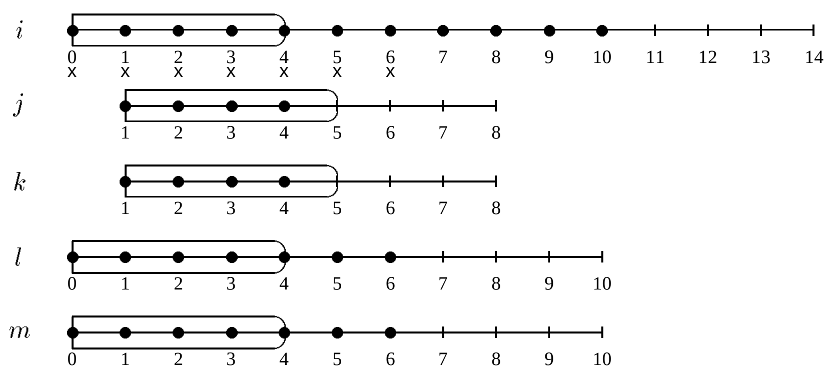

Figure 10 Five activities requiring one unit of a resource with capacity 2

the interval $[1, 9)$, while the interval consistency constraint is satisfied for all other intervals, including all activity or task intervals.

The example leads us back to the initial questions (1) how we should choose the hypothesis $\mathcal{H}$ and (2) for what time intervals the capacity constraint should be tested. Let us first address the second question.

The question has recently been answered by Schwindt (1997) and, independently, by Baptiste et al. (1998). They consider the difference between available and required resource capacity for a given time interval, i.e. the difference between the left and right side of the interval consistency constraint 2, as defined by the following slack function:

$$\text{slack}(\mathcal{V}_k, t_1, t_2) = R_k \cdot (t_2 - t_1) - W(\mathcal{V}_k, t_1, t_2). \tag{25}$$

By studying the possible extrema of the slack function for a given $\mathcal{V}_k$, the set of intervals $[t_1, t_2)$ can be characterised for which the slack function can take a local or global minimum and may thus violate an interval consistency constraint. Schwindt and Baptiste et al. have shown that the number of such minimum-slack intervals is of order of magnitude $O(|\mathcal{V}_k|^2)$ and have given a characterisation of the intervals (the one in Schwindt (1997) is slightly tighter). Thus, as we know from the initial example, the set of minimum-slack intervals is larger than the set of activity intervals but still of order $O(|\mathcal{V}_k|^2)$. Since an intuitive description of the minimum-slack intervals is hard to give, and because the proof is lengthy, we do not describe the set of minimum-slack intervals in more detail.

Baptiste et al. have developed an $O(|\mathcal{V}_k|^2)$ algorithm for testing the interval consistency constraint for all minimum-slack intervals, and an $O(|\mathcal{V}_k|^2 \log |\mathcal{V}_k|)$ algorithm has been described by Schwindt, who has used the interval consistency condition for deriving lower bounds for RCPSP/max instances.

In order to reduce activity domains, Baptiste et al. suggest to use hypothetical constraints of the type $st_i \leq t_x$, similar to the initial example. The time complexity of an algorithm that computes all domain reductions which can be obtained on the minimum-slack intervals follows from the fact that there are $|\mathcal{V}_k|$ activities to be tested and $O(|\mathcal{V}_k|^2)$ minimal-slack intervals. The development of a quadratic algorithm to compute all domain reductions is an open issue.

5.3 Unit-interval consistency

An important special case of the general interval consistency constraint 2 is obtained when we consider time intervals of width one. The following theorem describes a test for checking unit-intervals, based on the slack function defined in Equation 25.

Theorem 9 (Unit-width interval test). *Let $i \in \mathcal{V}_k$. If, for some time t,*

$$\textit{slack}(\mathcal{V}_k \setminus \{i\}, t, t+1) < r_{ik} \tag{26}$$

then the domain of i can be reduced in the following way:

$$\delta_i := \delta_i \setminus \{t' | t - p_i < t' < t + 1\}. \tag{27}$$

Proof. Obvious. □

The test is interesting because it is easy to design efficient algorithms for it. There is an obvious $O(T|\mathcal{V}_k|)$ algorithm for performing all reductions deducible by the test, where T is the time horizon of the problem. The test can also be efficiently implemented through tables reflecting remaining or used capacity over time, using either vector or support point representations of the capacity profile; whenever there is a change in the table for time t — caused by an update of some activity domain — the domain bounds of all activities $i \in \mathcal{V}_k$, for which $t \in \delta_i$, can be tested and updated with effort of at most $O(|\mathcal{V}_k|)$.

Tests equivalent or similar to the unit-interval consistency test have for instance been described in Le Pape (1994, 1995), Nuijten (1994), Caseau and Laburthe (1996), Klein and Scholl (1997).

For disjunctive scheduling, the unit-interval test is subsumed in the 2-consistency test.

5.4 Fully and partially elastic relaxations

This section describes two relaxations of the scheduling problem that have been suggested by Baptiste et al. (1998). The relaxations describe necessary conditions for the existence of a feasible schedule. They are based upon the idea of trying to answer the question whether there exists an integer function $es_k(t, i)$, for elastic schedule, that describes the number of work units assigned to all activities throughout their time windows so that for every activity the total number of units assigned equals the required work. The capacity assignment defined by $es_k(t, i)$ is elastic in the sense that it does not necessarily take the strict rectangular shape required in the original problem.

The fully elastic relaxation is the decision problem of deciding whether a function $es_k(t,i)$ exists such that the following constraints hold:

$$es_k(t,i) = 0, \quad \text{for all } i \in \mathcal{V}_k \text{ and } t \notin \delta_i \tag{28}$$

$$\sum_t es_k(t,i) = p_i r_{ik}, \quad \text{for all } i \in \mathcal{V}_k \tag{29}$$

$$\sum_{i \in \mathcal{V}_k} es_k(t,i) \leq R_k, \quad \text{for all } t. \tag{30}$$

A tighter relaxation can be obtained by adding the two following constraints.

$$\sum_{t'<t} es_k(t',i) \leq R_k \cdot (t - \min \delta_i), \quad \text{for all } i \text{ and } t \in \delta_i \tag{31}$$

$$\sum_{t \leq t'} es_k(t',i) \leq R_k \cdot (\max \delta_i - t), \quad \text{for all } i \text{ and } t \in \delta_i. \tag{32}$$

The resulting decision problem is called partially elastic relaxation; the way in which assigned work may float within the activity time window is more restricted than in the fully elastic case.

The partially and fully elastic relaxations can be used to deduce activity domain reductions in the usual way. If, after adding hypothetical constraint $\mathcal{H}$, it can be shown that no function $es_k(t,i)$ exists that satisfies Constraints 28–32, then $\neg\mathcal{H}$ must hold. Baptiste et al. describe an $O(|\mathcal{V}_k|^2)$ domain reduction algorithm based upon the fully elastic relaxation and an $O(\log|\{r_{ik}\}| \cdot |\mathcal{V}_k|^2)$ algorithm using the partially elastic relaxation, where $|\{r_{ik}\}|$ is the number of distinct resource capacity requirements.

The partially elastic relaxation is strictly weaker than the general interval consistency constraint (Baptiste et al. 1998).

6 Summary

We have presented a general, unifying framework for understanding interval capacity consistency tests. Within this framework, we have surveyed and extended previous results that have been obtained in the areas of Operations Research and Artificial Intelligence. We have related the concept of energetic reasoning to sequence consistency tests known under the names of immediate selection or edge finding.

The interval consistency tests described in this chapter have been applied frequently and with great success for solving disjunctive scheduling problems. Fewer and so far less conclusive results have been reported for the application of the tests for cumulative scheduling. Many of the tests that we have described are available in general purpose scheduling software libraries such as ILOG Scheduler (Le Pape 1994, 1995, Nuijten and Le Pape 1998), CHIP (Aggoun and Beldiceanu 1993), or CLAIRE Schedule (Le Pape and Baptiste 1996).

Acknowledgments

This research has been supported by the Deutsche Forschungsgemeinschaft under grant Pe 514/7 and by the Friedrich Ebert Stiftung.

References

AGGOUN, A. AND N. BELDICEANU. 1993. Extending CHIP in Order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling* **17**, 57–73.

APPLEGATE, D. AND W. COOK. 1991. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing* **3**, 149–156.

BAPTISTE, P. AND C. LE PAPE. 1995. A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling. In *Proceedings of the* 14^{th} *International Joint Conference on Artificial Intelligence*. Montreal.

BAPTISTE, P. AND C. LE PAPE. 1996. Edge-finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling. In *Proceedings of the* 15^{th} *Workshop of the U.K. Planning Special Interest Group*. Liverpool, UK.

BAPTISTE, P., C. LE PAPE AND W. P. NUIJTEN. 1998. Satisfiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems. Research report, Université de Technologie de Compiègne.

BARTUSCH, M., R. MÖHRING AND F. RADERMACHER. 1988. Scheduling Project Networks with Resource Constraints and Time windows. *Annals of Operations Research* **16**, 201–240.

BŁAŻEWICZ, J., W. DOMSCHKE AND E. PESCH. 1996a. The Job Shop Scheduling Problem: Conventional and New Solution Techniques. *European Journal of Operational Research* **93**, 1–33.

BŁAŻEWICZ, J., K. H. ECKER, E. PESCH, G. SCHMIDT AND J. WĘGLARZ. 1996b. *Scheduling Computer and Manufacturing Processes*. Springer, Berlin.

BŁAŻEWICZ, J., E. PESCH AND M. STERNA. 1997. Application of Modified Disjunctive Graphs for Job Shop Scheduling Problems. Working paper.

BŁAŻEWICZ, J., E. PESCH AND M. STERNA. 1998. A Branch and Bound Algorithm for the Job shop Scheduling Problem. In *Beyond Manufacturing Resource Planning (MRP II)*, A. Drexl and A. Kimms, eds. Springer, Berlin. To appear.

BRUCKER, P., A. DREXL, R. MÖHRING, K. NEUMANN AND E. PESCH. 1998. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. Working paper.

BRUCKER, P., B. JURISCH AND A. KRÄMER. 1996a. The Job-Shop Problem and Immediate Selection. *Annals of Operations Research* **50**, 73–114.

BRUCKER, P., A. SCHOO AND O. THIELE. 1996b. A Branch and Bound Algorithm for the Resource-Constrained Project Scheduling Problem. Osnabrücker Schriften zur Mathematik 178, Universität Osnabrück.

CARLIER, J. 1982. The One-Machine Sequencing Problem. *European Journal of Operational Research* **11**, 42–47.

CARLIER, J. AND E. PINSON. 1989. An Algorithm for Solving the Job-Shop Problem. *Management Science* **35**, 164–176.

CARLIER, J. AND E. PINSON. 1990. A Practical Use of Jackson's Preemptive Schedule for the Job Shop Problem. *Annals of Operations Research* **26**, 269–287.

CARLIER, J. AND E. PINSON. 1994. Adjustments of Heads and Tails for the Job-Shop Problem. *European Journal of Operational Research* **78**, 146–161.

CASEAU, Y. AND F. LABURTHE. 1994. Improved CLP Scheduling with Task Intervals. In *Proceedings of the* 11^{th} *International Conference on Logic Programming*, P. van Hentenryck, ed. MIT-Press.

CASEAU, Y. AND F. LABURTHE. 1995. Disjunctive Scheduling with Task Intervals. Tech. Rep. 95-25, Laboratoire d'Informatique de l'Ecole Normale Supérieure Paris.

CASEAU, Y. AND F. LABURTHE. 1996. Cumulative Scheduling with Task Intervals. In *Proceedings of the Joint International Conference on Logic Programming*. MIT-Press.

DOMSCHKE, W. AND A. DREXL. 1991. Kapazitätsplanung in Netzwerken: Ein Überblick über neuere Modelle und Verfahren. *OR Spektrum* **13**, 63–76.

FISHER, H. AND G. THOMPSON. 1963. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In *Industrial Scheduling*, J. Muth and G. Thompson, eds. Prentice-Hall, Englewood Cliffs, NF.

HEILMANN, R. AND C. SCHWINDT. 1997. Lower Bounds for RCPSP/max. Tech. Rep. WIOR-511, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

JACKSON, J. 1956. An Extension of Johnson's Results on Job Lot Scheduling. *Naval Research Logistics Quarterly* **3**, 201–203.

KLEIN, R. AND A. SCHOLL. 1997. Computing Lower Bounds by Destructive Improvement — an Application to Resource-Constrained Project Scheduling. Schriften zur Quantitativen Betriebswirtschaftslehre 4/97, Technische Hochschule Darmstadt.

KOLISCH, R., A. SPRECHER AND A. DREXL. 1995. Characterization and Generation of a General Class of Resource-constrained Project Scheduling Problems. *Management Science* **41**, 1693–1703.

KUMAR, V. 1992. Algorithms for Constraint-Satisfaction Problems: A Survey. *A.I. Magazine* **13**, 32–44.

LE PAPE, C. 1994. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering* **3**, 55–66.

LE PAPE, C. 1995. Three Mechanisms for Managing Resource Constraints in a Library for Constraint-Based Scheduling. In *Proceedings of the INRIA/IEEE Conference on Emerging Technologies and Factory Automation*. Paris.

LE PAPE, C. AND P. BAPTISTE. 1996. A Constraint Programming Library for Preemptive and Non-Preemptive Scheduling. In *Proceedings of the* 12^{th} *European Conference on Aritificial Intelligence*.

LOPEZ, P. 1991. *Aproche énergétique pour l'ordonnancement de tâches sous contraintes te temps et de ressources*. Ph.D. thesis, Université Paul Sabatier, Toulouse. Cited after Lopez et al. 1992.

LOPEZ, P., J. ERSCHLER AND P. ESQUIROL. 1992. Ordonnancement de tâches sous contraintes: une approche énergétique. *RAIRO Automatique, Productique, Informatique Industrielle* **26**, 453–481.

MARTIN, P. AND D. B. SHMOYS. 1996. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. In *Proceedings of the* 5^{th} *International IPCO Conference*.

NUIJTEN, W. P. 1994. *Time and Resource Constrained Scheduling: A Constraint*

Satisfaction Approach. Ph.D. thesis, Eindhoven University of Technology.

NUIJTEN, W. P. AND E. AARTS. 1996. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling. *European Journal of Operational Research* **90**, 269–284.

NUIJTEN, W. P. AND C. LE PAPE. 1998. Constraint-based Job Shop Scheduling with ILOG SCHEDULER. *Journal of Heuristics* **3**, 271–286.

PESCH, E. AND U. TETZLAFF. 1996. Constraint Propagation Based Scheduling of Job Shops. *INFORMS Journal on Computing* **8**, 144–157.

SCHWINDT, C. 1997. *Verfahren zur Lösung des Ressourcenbeschränkten Projektdauerminimierungsproblems mit Planungsabhängigen Zeitfenstern*. Ph.D. thesis, Universität Fridericiana zu Karlsruhe.

TSANG, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press, London.

VAN HENTENRYCK, P. 1989. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA.