16 Constraint Programming and Disjunctive Scheduling

Constraint propagation is an elementary method for reducing the search space of combinatorial search and optimization problems which has become more and more important in the last decades. The basic idea of constraint propagation is to detect and remove inconsistent variable assignments that cannot participate in any feasible solution through the repeated analysis and evaluation of the variables, domains and constraints describing a specific problem instance.

This chapter is based on Dorndorf et al. [DPP00] and its contribution is twofold. The first contribution is a description of efficient constraint propagation methods also known as consistency tests for the disjunctive scheduling problem (DSP) which is a generalization of the classical job shop scheduling problem (JSP). By applying an elementary constraint based approach involving a limited number of search variables, we will derive consistency tests that ensure 3-*b*-consistency. We will further present and analyze both new and classical consistency tests which to some extent are generalizations of the aforementioned consistency tests involving a higher number of variables, but still can be implemented efficiently with a polynomial time complexity. Further, the concepts of energetic reasoning and shaving are analyzed and discussed.

The other contribution is a classification of the consistency tests derived according to the domain reduction achieved. The particular strength of using consistency tests is based on their repeated application, so that the knowledge derived is propagated, i.e. reused for acquiring additional knowledge. The deduction of this knowledge can be described as the computation of a fixed point. Since this fixed point depends upon the order of the application of the tests, we first derive a necessary condition for its uniqueness. We then develop a concept of dominance which enables the comparison of different consistency tests as well as a simple method for proving dominance. An extensive comparison of all consistency tests is given. Quite surprisingly, we will find out that some apparently stronger consistency tests are subsumed by apparently weaker ones. At the same time an open question regarding the effectiveness of energetic reasoning is answered.

16.1 Introduction

Exact solution methods for solving combinatorial search and optimizations problems generally consist of two components: (a) a search strategy which organizes the enumeration of all potential solutions and (b) a search space reduction strate-

J. Blazewicz et al., *Handbook on Scheduling*, International Handbooks on Information Systems, https://doi.org/10.1007/978-3-319-99849-7_16

gy which diminishes the number of potential solutions. However, due to the exponentially growing size of the search space, even an intelligent organization of the search will eventually fail, so that only the application of efficient search space reduction mechanisms will allow the solution of more difficult problems. Consequently, as an elementary method of search space reduction, constraint propagation has become more and more important in the last decades. Constraint propagation has its origins in the popular field of constraint programming which models combinatorial search problems as special instances of the *constraint satisfaction problem* (CSP) . The basic idea of constraint propagation is to evaluate implicit constraints through the repeated analysis of the variables, domains and constraints that describe a specific problem instance. This analysis makes it possible to detect and remove *inconsistent* variable assignments that cannot participate in any solution by a merely partial problem analysis.

One of our main objectives is to present and derive efficient constraint propagation techniques also known as consistency tests for the *disjunctive scheduling problem* (DSP) which is a generalization of the classical job shop scheduling problem (JSP). The DSP constitutes a perfect object of study due to the trade-off between its computational complexity and its simple description. On the one hand, within the class of NP-hard problems the DSP has been termed to be one of the most intractable problems. This view is best supported by the notorious 10×10 problem instance of the JSP introduced by Muth and Thompson [MT63] which resisted any solution attempts for several decades and was only solved more than 25 years later by Carlier and Pinson [CP89]. On the other hand, the disjunctive model introduced by Roy and Sussman [RS64] provides an illustrative and simple representation of the DSP which is only based on two types of constraints which in scheduling are known as *precedence* and *disjunctive constraints*.

An elementary analysis of the DSP involving a limited number of search variables derives the consistency tests that ensure 3-*b*-consistency. These consistency tests can be generalized and, although their application does not establish a higher level of consistency, they enable powerful domain reductions in polynomial time. Notice, that establishing n-consistency for any n is NP-hard, thus the existence of a polynomial algorithm is not very probable. Furthermore the concepts of energetic reasoning and shaving are presented.

The other objective of this chapter is a classification of the consistency tests derived according to the domain reduction achieved. A new dominance criterion that allows a comparison of consistency tests in the aforementioned sense and simple methods for proving dominance are presented. An extensive study of all consistency tests is given. Quite surprisingly, comparing the extent of the search space reduction induced, we will find out that some apparently stronger consistency tests are subsumed by apparently weaker ones.

The remainder of this chapter is organized as follows. Section 16.2 introduces the CSP. Several concepts of consistency are proposed which may serve as a theoretical basis for constraint propagation techniques. We define consistency tests and present the aforementioned dominance criterion for comparing them.

Section 16.3 describes the DSP and examines its relation to the CSP. Section 16.4 extensively describes constraint propagation techniques for the DSP. Notice that although we focus on the basic DSP, the results of this work also apply in an unchanged manner to some important extensions of the DSP, for instance, the DSP with release times and due dates. Section 16.5 finally summarizes the results.

16.2 Constraint Satisfaction

Search and optimization problems such as the disjunctive scheduling problem are generally modelled as special subclasses of the *constraint satisfaction problem* (CSP) or the *constraint optimization problem* (COP) . We will give a short introduction to these problem classes in subsection 16.2.1. In subsection 16.2.2 we will then describe constraint propagation methods and different concepts of consistency.

16.2.1 The Constraint Satisfaction and Optimization Problem

The CSP can be roughly described as follows: "Given a domain specification, find a solution x, such that x is a member of a set of possible solutions and it satisfies the problem conditions" [Ama70]. The COP additionally requires that the solution found optimizes some objective function.

The CSP was first formalized and studied by Huffman [Huf71], Clowes [Clo71] and Waltz [Wal75] in vision research for solving line-labelling problems. Haralick and Shapiro [HS79, HS80] and Mackworth [Mac92] discuss general algorithms and applications of CSP solving. Van Hentenryck [Hen92] and Cohen [Coh90] tackle the CSP from a constraint logic programming viewpoint. Comprehensive overviews on the CSP are provided by Meseguer [Mes89] and Kumar [Kum92]. An exhaustive study of the theory of constraint satisfaction and optimization can be found in [Tsa93]. We will only present the necessary aspects and start with some basic definitions.

The *domain* of a variable is the set of all values that can be assigned to the variable. We will assume in this section that domains are finite and later allow for infinite but discrete domains. The domain associated with the variable x is denoted by $\mathcal{D}(x)$. If $\mathcal{V} = \{x_1, \cdots, x_n\}$ is a set of variables and $\mathcal{DOM} = \{\mathcal{D}(x_1), \cdots, \mathcal{D}(x_n)\}$ the set of domains, then an *assignment* $a = \{a_1, \cdots, a_n\}$ is an element of the Cartesian product $\mathcal{D}(x_1) \times \cdots \times \mathcal{D}(x_n)$; in other words, an assignment instantiates each variable x_i with a value $a_i \in \mathcal{D}(x_i)$ from its domain.

A *constraint* c on $\mathcal{DOM}$ is a function $c\colon \mathcal{D}(x_{i_1}) \times \cdots \times \mathcal{D}(x_{i_k}) \rightarrow \{\text{true}, \text{false}\}$, where $\mathcal{V}' := \{x_{i_1}, \cdots, x_{i_k}\}$ is a non empty set of variables. The cardinality $|\mathcal{V}'|$ is also called the arity of c. If $|\mathcal{V}'| = 1$ or $|\mathcal{V}'| = 2$ then we speak of unary and bina-

ry constraints respectively. An assignment $a = \mathcal{D}(x_1) \times \ldots \times \mathcal{D}(x_n)$ satisfies c iff $c(a_{i_1}, \cdots, a_{i_k}) = \text{true}$.

Definition 16.2.1

An instance I of the *constraint satisfaction problem* (CSP) is defined by a tuple $I = (\mathcal{V}, \mathcal{DOM}, \mathcal{CONS})$, where $\mathcal{V}$ is a finite set of variables, $\mathcal{DOM}$ the set of associated domains and $\mathcal{CONS}$ a finite set of constraints on $\mathcal{DOM}$. An assignment a is *feasible* iff it satisfies all constraints in $\mathcal{CONS}$. A feasible assignment is also called a *solution* of I. We denote with $\mathcal{F}(I)$ the set of all feasible assignments (solutions) of I.

Given an instance I of the CSP, the associated problem is to find a solution $a \in \mathcal{F}(I)$ or to prove that I has no solution.

As distinguished from the constraint satisfaction problem, the constraint optimization problem searches for a solution which optimizes a given objective function. We will only consider the case of minimization, as maximization can be handled symmetrically.

Definition 16.2.2

An instance of the *constraint optimization problem* (COP) is defined by a tuple $I = (\mathcal{V}, \mathcal{DOM}, \mathcal{CONS}, z)$, where $(\mathcal{V}, \mathcal{DOM}, \mathcal{CONS})$ is an instance of the CSP and z an objective function $z : \mathcal{D}(x_1) \times \ldots \times \mathcal{D}(x_n) \rightarrow \mathbb{R}$. Defining

$$z_{min}(I) := \begin{cases} \min\limits_{b \in \mathcal{F}(I)} z(b) & \text{if } \mathcal{F}(I) \neq \emptyset, \\ \infty & \text{otherwise,} \end{cases}$$

an assignment a is called an *optimal solution* of I iff a is feasible and $z(a) = z_{min}(I)$.

Given an instance I of the COP, the associated problem is to find an optimal solution of I and to determine $z_{min}(I)$.

It is not hard to see that the CSP and the COP are intractable and belong to the class of NP-hard problems (c.f. Section 2.2).

An instance of the CSP can be represented by means of a graph (*constraint graph*) which visualizes the interdependencies between variables that are induced by the constraints. If we restrict our attention to unary and binary constraints then the definition of a constraint graph G is quite straightforward. The vertex set of G corresponds to the set of all variables $\mathcal{V}$, while the edge set is defined as follows: two vertices $x_i, x_j \in \mathcal{V}$, $i \neq j$, are connected by an undirected edge iff there exists a constraint $c(x_i, x_j) \in \mathcal{CONS}$. This can be generalized to constraints of arbitrary arity using the notion of hypergraphs [Tsa93]. Figure 16.2.1 shows a typical CSP instance and the corresponding constraint graph.

16.2.2 Constraint Propagation

From a certain point of view, the CSP and the COP are quite simple problems. Since we assumed that the domains of a CSP instance I are finite which for most interesting problems is not a serious restriction, I can be solved by a simple generate-and-test algorithm that works as follows: enumerate all assignments $a \in \mathcal{D}(x_1) \times \cdots \times \mathcal{D}(x_n)$ and verify whether a satisfies all constraints $c \in \mathcal{CONS}$; stop if the answer is "yes". The COP can be solved by enumerating all feasible assignments and storing the one with minimal objective function value.

Unfortunately, this method is not practicable due to the size of the search space which grows exponentially with the number of variables. In the worst case, all assignments of a CSP instance have to be tested which cannot be carried out efficiently except for problem instances too small to be of any practical value. Thus, it suggests itself to examine methods which reduce the search space prior to starting (or during) the search process.

One such method of search space reduction which only makes use of simple inference mechanisms and does not rely on problem specific knowledge is known as constraint propagation. The origins of constraint propagation go back to Waltz [Wal72] who more than three decades ago developed a now well-known filtering algorithm for labelling three-dimensional line diagrams.

The basic idea of constraint propagation is to make implicit constraints more visible through the repeated analysis and evaluation of the variables, domains and constraints describing a specific problem instance. This makes it possible to detect and remove inconsistent variable assignments that cannot participate in any solution by a merely partial problem analysis.

Two complexity related problems arise when performing constraint propagation. One problem depends upon the number of variables and constraints that are examined simultaneously, while the other problem is caused by the size of the domains. These problems are usually tackled by limiting the number of variables and constraints (local consistency with respect to all subsets of k variables) and the number of domain assignments (domain- or d-consistency, bound- or b-consistency) that are considered in the examination. These different concepts will be discussed further below. We start with some simple examples, as this is the easiest way to introduce constraint propagation.

Example 16.2.3

Let $I = (\mathcal{V}, \mathcal{DOM}, \mathcal{CONS})$ be the CSP instance shown in Figure 16.2.1. A simple analysis of the constraints (*i*) to (*vi*) allows us to reduce the domains of the variables x_1, x_2 and x_3. We distinguish between the domains $\mathcal{D}(x_i)$ and the reduced domains $\delta(x_i)$. At the beginning, of course, $\delta(x_i) = \mathcal{D}(x_i)$ for $i \in \{1,2,3\}$.

$\mathcal{V} = \{x_1, x_2, x_3\}$,
$\mathcal{D}(x_1) = \{1,\cdots,10\}$,
$\mathcal{D}(x_2) = \{1,\cdots,10\}$,
$\mathcal{D}(x_3) = \{1,\cdots,10\}$,
(i) $1 \le x_1 \le 4$,
(ii) $1 \le x_2 \le 4$,
(iii) $1 \le x_3 \le 4$,
(iv) $x_1 + x_2 = 4$,
(v) $x_1 + x_3 = 5$,
(vi) $x_2 + x_3 \ge 6$.

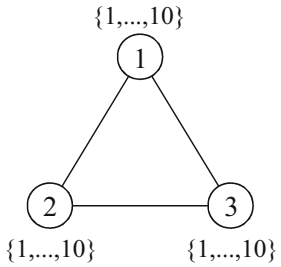


Figure 16.2.1 *Example* 16.2.3.

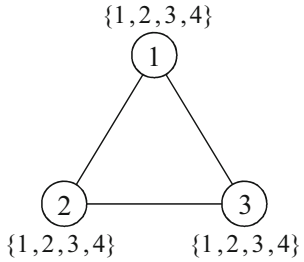


Figure 16.2.2 *Step* 1.

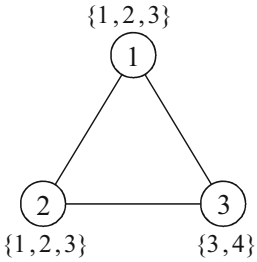


Figure 16.2.3 *Steps* 2, 3 *and* 4.

1. The unary constraints (i) - (iii) yield the trivial but considerable reduction $\delta(x_1) := \delta(x_2) := \delta(x_3) := \{1,2,3,4\}$ (see Figure 16.2.2).
2. We next examine pairs of variables. Let us start with the pair (x_1,x_2) and the constraint (iv). If we choose, for instance, the assignment $a_1 = 4$ then there obviously exists no assignment $a_2 \in \delta(x_2) = \{1,\cdots,4\}$ which satisfies (iv) $x_1 + x_2 = 4$. Hence, the value 4 can be removed from $\delta(x_1)$. The same argument is not applicable to $a_1 = 1,2,3$, so we currently can only deduce $\delta(x_1) := \{1,2,3\}$.
3. Since (iv) is symmetric in x_1 and x_2, we can as well set $\delta(x_2) := \{1,2,3\}$.
4. Consider now the pair (x_2,x_3) and constraint (vi). As $a_2 \in \{1,2,3\}$, i.e. $a_2 \le 3$, the constraint (vi), $x_2 + x_3 \ge 6$, is only satisfied for $a_3 \ge 3$. We therefore obtain $\delta(x_3) := \{3,4\}$ (see Figure 16.2.3).
5. Now let us turn to the pair (x_1,x_3) and constraint (v). Since $a_3 = 3$ or $a_3 = 4$, constraint (v), $x_1 + x_3 = 5$, yields $a_1 \ne 3$, and we can set $\delta(x_1) := \{1,2\}$.
6. Finally, studying constraint (iv) once more, we can remove $a_2 = 1$ and set

$\delta(x_2) := \{2,3\}$ (see Figure 16.2.4).

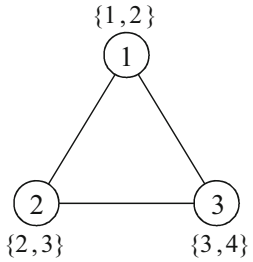


Figure 16.2.4 *Steps* 5 *and* 6.

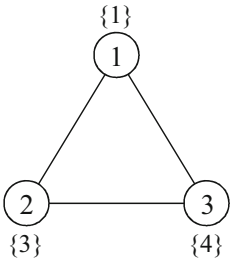


Figure 16.2.5 *The final step.*

At this point, no more values can be excluded from the current domains through the examination of pairs of variables. If we stop propagation now then the search space reduction is already of a considerable size. Prior to our simple analysis, the search space was of cardinality $|\mathcal{D}(x_1) \times \mathcal{D}(x_2) \times \mathcal{D}(x_3)| = 10 \cdot 10 \cdot 10 = 1000$, afterwards the cardinality dropped down to $|\delta(x_1) \times \delta(x_2) \times \delta(x_3)| = 2 \cdot 2 \cdot 2 = 8$.

Extending our analysis to triples of variables reduces the search space even more. Given, for instance, $a_1 = 2$, constraint (*iv*) implies $a_2 = 2$, while (*v*) implies $a_3 = 3$. Since $a_2 + a_3 = 5 < 6$, this is a contradiction to the constraint (*vi*). Reducing $\delta(x_1)$ to $\{1\}$, we can immediately deduce $\delta(x_2) = \{3\}$ and $\delta(x_3) = \{4\}$ which is shown in Figure 16.2.5. Hence, only the assignment $a = (1,3,4)$ is feasible and $\mathcal{F}(I) = \{(1,3,4)\}$ is the solution space of I. □

Example 16.2.4

Consider now the CSP instance $I = (\mathcal{V}, \mathcal{DOM}, \mathcal{CONS})$ shown in Figure 16.2.6. Here, the constraint $a \bmod b = c$ yields true, if a divided by b has a remainder of c. It is possible to show that this CSP instance has eight feasible solutions:

$$\mathcal{F}(I) = \{(4,7,5),(4,7,10),(5,6,1),(5,6,6),(9,2,5),(9,2,10),(10,1,1),(10,1,6)\}$$

$\mathcal{V} = \{x_1, x_2, x_3\}$,
$\mathcal{D}(x_1) = \{1,\cdots,10\}$,
$\mathcal{D}(x_2) = \{1,\cdots,10\}$,
$\mathcal{D}(x_3) = \{1,\cdots,10\}$,
(*i*) $(x_1 + x_2) \bmod 10 = 1$,
(*ii*) $(x_1 \cdot x_3) \bmod 5 = 0$,
(*iii*) $(x_2 + x_3) \bmod 5 = 2$.

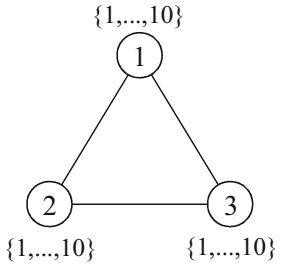


Figure 16.2.6 *Example* 16.2.4.

However, finding these solutions using only constraint propagation is not as easy

as in Example 16.2.3. It is not hard to see that the corresponding current domains $\delta(x_1)$, $\delta(x_2)$ and $\delta(x_3)$ cannot be reduced by examining pairs of variables. Consider, for instance, the pair (x_1,x_2) and constraint (*i*): for each assignment $a_1 \in \delta(x_1)$, there exists an assignment $a_2 \in \delta(x_2)$ such that (*i*) is satisfied. Similar conclusions can be drawn if the roles of x_1 and x_2 are interchanged or if we study the pairs (x_2, x_3) and (x_1, x_3).

To derive further information, we have to examine pairs of assignments. We may, for instance, find out that the assignments $\{1\} \times \{1,\cdots,9\}$ of the variables x_1 and x_2 cannot participate in any feasible solution, since they do not satisfy constraint (*i*). Thus given $a_1 = 1$, the only interesting assignment is $a_2 = 10$. Similar results can be obtained for $a_1 = 2$, etc. This analysis, however, increases the overhead in terms of computational complexity and storage capacity considerably, since pairs of assignments have to be dealt with, and it is not clear at all whether this additional overhead can be offset by the search space reduction achieved. □

These examples demonstrate that constraint propagation can be quite powerful, reducing the search space of a "favourable" CSP instance to a great extent after a few steps of propagation. In the worst case, however, constraint propagation does not yield a substantial reduction of the search space and even slows down the complete solution process due to the additional computations. In general, the outcome of constraint propagation lies between these two extremes: some but not all infeasible solutions can be discarded if constraint propagation is restricted to techniques which can be implemented efficiently. Thus, constraint propagation complements, but does not replace a systematic search.

After this intuitive introduction to constraint propagation, it is now necessary to provide a theoretical environment which allows us to design and assess constraint propagation techniques. We have informally described constraint propagation as "the reduction of the search space of a CSP instance through the analysis of variables, domains and constraints". The question how far this reduction should be carried out, we would readily answer "as far as possible". Remember, however, that any CSP instance is uniquely determined through its variables, domains and constraints. Thus, if we took this description literally then constraint propagation would just be a synonym to *solving* the CSP which of course is not sensible, because we initially have introduced constraint propagation in order to *simplify* the solution of the CSP. Further, we already have seen that constraint propagation is only useful up to a certain extent due to an increasing computational complexity. We therefore present different concepts of consistency which may serve as a theoretical basis for propagation techniques. Roughly speaking, a concept of consistency defines the maximal search space reduction that is possible regarding some specific criteria.

k-Consistency

The first concepts of consistency have been presented in the early seventies by Montanari [Mon74], who introduced the notions of *node-*, *arc-* and *path-consistency*. Roughly speaking, these concepts are based on the examination of constraints containing k variables, where $k = 1, 2, 3$, with their names being derived from the representation of a CSP instance as a constraint graph. Notice, that in the last section examples have been given of how to achieve node- and arc-consistency which will be seen more clearly further below. These concepts of consistency have been generalized by Freuder [Fre78] in a natural manner to the notion of *k-consistency*. For a detailed analysis of k-consistency see for instance [Tsa93]. We will only describe the basic ideas in an informal way.

In order to define k-consistency we have to introduce the notion of *k-feasibility*. Let $a = (a_1, \cdots, a_n)$ be an assignment of a given CSP instance. A partial assignment of k variables $(a_{i_1}, \cdots, a_{i_k})$ is *k-feasible*, if it satisfies all constraints which contain these variables only (or any subset of them). The motivation of the definition of k-consistency is based on the following observation: a can only be feasible, if for a given k any partial assignment $(a_{i_1}, \cdots, a_{i_k})$ is k-feasible. Inversely, any partial assignment of k variables, that is not feasible, is not interesting and hints at an inconsistent state.

In Freuder's words [Fre78] k-consistency is achieved if for any $(k-1)$-feasible assignment of $k-1$ variables (taken from a set $\delta(x_{i_1}, \cdots, x_{i_{k-1}}) \subseteq \mathcal{D}(x_{i_1}) \times \cdots \times \mathcal{D}(x_{i_{k-1}})$) and any choice of a k^{th} variable, there exists an assignment of the k^{th} variable (taken from a set $\delta(x_{i_k}) \subseteq \mathcal{D}(x_{i_k})$), such that the assignment of the k variables taken together is k-feasible.

Note that the *property* of k-consistency is always relative to the sets $\delta(x_{i_1}, \cdots, x_{i_{k-1}})$ and $\delta(x_{i_k})$. Thus, in order to *establish* k-consistency, starting from an inconsistent state, this implicitly requires a $(k-1)$-dimensional administration of these sets. At the beginning, these sets contain all assignments, that is, $\delta(x_{i_1}, \cdots, x_{i_{k-1}}) := \mathcal{D}(x_{i_1}) \times \cdots \times \mathcal{D}(x_{i_{k-1}})$ and $\delta(x_{i_k}) := \mathcal{D}(x_{i_k})$. Inconsistent assignments are then eventually discarded, until k-consistency is reached.

1-consistency is quite easy to achieve: if $x_i \in \mathcal{V}$ is a variable and $c(x_i)$ is a unary constraint then all assignments $a_i \in \delta(x_i)$ for which $c(a_i) =$ false are removed. In order to establish 2-consistency, pairs of variables $x_i, x_j \in \mathcal{V}$ and binary constraints $c(x_i, x_j)$ have to be examined: an assignment $a_i \in \delta(x_i)$ can be removed if $c(a_i, a_j) =$ false for all $a_j \in \delta(x_j)$. Analogously, 3-consistency requires the examination of triples of variables $x_i, x_j, x_k \in \mathcal{V}$ and removes pairs of assignments $(a_i, a_j) \in \delta(x_i, x_j)$, etc. As already mentioned, 1- and 2-consistency coincide with the notions of node- and arc-consistency, whereas 2- and 3-consistency taken together are equivalent to path-consistency, see e.g. [Mon74,

Mac77, MH86, Tsa93]. 1-, 2- and 3-consistency have also been summarized under the name of *lower-level* consistency as opposed to *higher-level* consistency, since only small subsets of variables, domains and constraints are evaluated simultaneously.

Efficient algorithms for establishing 1-, 2- and 3-consistency and an analysis of their complexity have been presented, among others, by Montanari [Mon74], Mackworth [Mac77], Mackworth and Freuder [MF85], Mohr and Henderson [MH86], Dechter and Pearl [DP88], Han and Lee [HL88], Cooper [Coo89] and Van Hentenryck et al. [HDT92]. Improved arc consistency algorithms AC-6 and AC-7 have been presented by Bessière [Bes94] and by Bessière et al. [BFR99]. Chen [Che99] has proposed a new arc consistency algorithm, AC-8, which requires less computation time and space than AC-6 and AC-7. Cooper developed an optimal algorithm which achieves k-consistency for arbitrary k [Coo89]. Jeavons et al.[JCC98] have identified a number of constraint classes for which some fixed level of local consistency is sufficient to ensure global consistency. They characterize all possible constraint types for which strong k-consistency guarantees global consistency, for each $k \geq 2$. Other methods for solving the CSP through the sole application of constraint propagation (*solution synthesis*) have been proposed by Freuder [Fre78], Seidel [Sei81] and Tsang and Foster [TF90]. The deductive approach proposed by Bibel [Bib88] is closely related to solution synthesis.

Domain-Consistency

Cooper's optimal algorithm [Coo89] for achieving k-consistency requires testing all subsets $\delta(x_{i_1},\cdots,x_{i_{k-1}}) \subseteq \mathcal{D}(x_{i_1}) \times\cdots\times \mathcal{D}(x_{i_{k-1}})$ of $(k-1)$-feasible assignments which is only practicable for small values of k. We therefore describe two weaker concepts of consistency.

The first concept is based on only storing the 1-dimensional sets $\delta(x_i) \subseteq \mathcal{D}(x_i)$ for all variables $x_i \in \mathcal{V}$. For reasons near at hand, $\delta(x_i)$ is also called the *current domain* of x_i. Intuitively, we can at most discard all values $a_i \in \delta(x_i)$ for which there exist no assignments $a_j \in \delta(x_j)$, $j \neq i$, such that $(a_1,\cdots,a_i,\cdots,a_n)$ is feasible. Alternatively, the feasibility condition can be replaced with the sufficient condition of k-feasibility which leads to a lower level of consistency. We refer to this concept of consistency as *domain-consistency* or *k-d-consistency*. Domain-consistency has been used, among others, by Nuijten [Nui94]. Formal definitions are provided below.

Definition 16.2.5

Let $I = (\mathcal{V},\mathcal{DOM},\mathcal{CONS})$ be an instance of the CSP. If $\delta(x_i) \subseteq \mathcal{D}(x_i)$ is the current domain of the variable $x_i \in \mathcal{V}$ then $\delta(x_i)$ is *complete* iff, for all feasible assignments $a = (a_1,\cdots,a_n)$, the value a_i is contained in $\delta(x_i)$.

Definition 16.2.6

Let $I = (\mathcal{V},\mathcal{DOM},\mathcal{CONS})$ be an instance of the CSP and $\Delta := \{ \delta(x_i) \mid x_i \in \mathcal{V} \}$ be the set of current domains, so that $\delta(x_i) \subseteq \mathcal{D}(x_i)$ is complete[1].

1. Δ is *k-d-consistent* for $1 \le k \le n$ iff, for all subsets $\mathcal{V}' := \{x_{i_1},\cdots,x_{i_{k-1}}\}$ of $k-1$ variables and any k^{th} variable $x_{i_k} \notin \mathcal{V}'$, the following condition holds:
$$\forall\, a_{i_k} \in \delta(x_{i_k}), \exists\, a_{i_1} \in \delta(x_{i_1}),\cdots, \exists\, a_{i_{k-1}} \in \delta(x_{i_{k-1}}) : (a_{i_1},\cdots,a_{i_k}) \text{ is } k\text{-feasible}.$$
2. Δ is *strong k-d-consistent* for $1 \le k \le n$ iff Δ is k'-d-consistent for all $1 \le k' \le k$.

The following naive algorithm establishes k-d-consistency: start with $\delta(x_i) := \mathcal{D}(x_i)$ for all $x_i \in \mathcal{V}$; choose variable x_{i_k} and assignment $a_{i_k} \in \delta(x_{i_k})$; test whether there exists a subset of $k-1$ variables $\mathcal{V}' := \{x_{i_1},\cdots,x_{i_{k-1}}\}$ which does not contain x_{i_k}, so that $(a_{i_1},\cdots,a_{i_{k-1}},a_{i_k})$ is not k-feasible for all $a_{i_1} \in \delta(x_{i_1}), \cdots, a_{i_{k-1}} \in \delta(x_{i_{k-1}})$; if the answer is "yes" then remove the assignment a_{i_k} from $\delta(x_{i_k})$; repeat this process with other assignments and/or variables until no more domain reductions are possible.

Example 16.2.7

Let us reconsider Example 16.2.4. After establishing n-d-consistency, the reduced domains $\delta(x_i)$ contain only assignments $a_i \in \mathcal{D}(x_i)$ for which there exists a feasible solution $(a_1,a_2,a_3) \in \mathcal{F}(I)$. Since the solution space is

$$\mathcal{F}(I) = \{(4,7,5), (4,7,10), (5,6,1), (5,6,6), (9,2,5), (9,2,10), (10,1,1), (10,1,6)\}$$

we obtain $\delta(x_1) = \{4,5,9,10\}$, $\delta(x_2) = \{1,2,6,7\}$, and $\delta(x_3) = \{1,5,6,10\}$. After the reduction, the search space is of size $|\,\delta(x_1) \times \delta(x_2) \times \delta(x_3)\,| = 4 \cdot 4 \cdot 4 = 64$ as compared to the original search space of size $|\,\mathcal{D}(x_1) \times \mathcal{D}(x_2) \times \mathcal{D}(x_3)\,| = 10 \cdot 10 \cdot 10 = 1000$ which is considerably larger. □

This gives us an indication of the maximal search space reduction that is possible if a solely domain oriented approach is chosen. Notice, however, that we did not yet discuss how to establish n-d-consistency other than to apply the naive algorithm, so an important question is whether there exists an efficient implementation after all. Before we deal with this issue, however, we will first present another concept of consistency.

---

[1] The completeness property which is usually omitted in other definitions of consistency ensures that no feasible solutions are removed. Without this property, $\Delta := \{\varnothing, \ldots, \varnothing\}$ would be *n-d*-consistent which obviously is not intended.

Bound-Consistency

Storing all values of the current domains $\delta(x_1),\cdots,\delta(x_n)$ still might be too costly. An interval oriented encoding of $\delta(x_i)$ provides an alternative if $\mathcal{D}(x_i)$ is totally ordered, for instance, if $\mathcal{D}(x_i) \subseteq \mathbb{N}_0$. In this case, we can identify $\delta(x_i)$ with the interval $\delta(x_i) := [l_i, r_i] := \{l_i, l_i + 1, \cdots, r_i - 1, r_i\}$, so that only the "left" and "right" bounds of $\delta(x_i)$ have to be stored. Therefore, this concept of consistency is usually referred to as *bound-consistency* or *k-b-consistency*. Bound-consistency has been discussed, among others, by Moore [Moo66], Davis [Dav87], van Beek [Bee92] and Lhomme [Lho93].

Definition 16.2.8 (*k-b-consistency*).

Let $I = (\mathcal{V},\mathcal{DOM},\mathcal{CONS})$ be an instance of the CSP and $\Delta := \{\, \delta(x_i) \mid x_i \in \mathcal{V} \,\}$ be the set of current domains, so that $\delta(x_i) \subseteq \mathcal{D}(x_i)$ is complete.

1. Δ is *k-b-consistent* for $1 \le k \le n$ iff, for all subsets $\mathcal{V}' := \{x_{i_1},\cdots,x_{i_{k-1}}\}$ of $k-1$ variables and any k^{th} variable $x_{i_k} \notin \mathcal{V}'$, the following condition holds:

$$\forall\, a_{i_k} \in \{l_{i_k}, r_{i_k}\}, \exists\, a_{i_1} \in \delta(x_{i_1}),\cdots,\exists\, a_{i_{k-1}} \in \delta(x_{i_{k-1}}) :$$

$$(a_{i_1},\cdots,a_{i_k}) \text{ is } k\text{-feasible}.$$

2. Δ is *strongly k-b-consistent* for $1 \le k \le n$ iff Δ is k'-*b*-consistent for all $1 \le k' \le k$.

A naive algorithm for establishing *k-b*-consistency is obtained by slightly modifying the naive *k*-d-consistency algorithm: instead of choosing $a_{i_k} \in \delta(x_{i_k})$, we may only choose (and remove) $a_{i_k} \in \{l_{i_k}, r_{i_k}\}$.

As a negative side effect, only the bounds l_i and r_i, but no intermediate value $l_i < a_i < r_i$ can be discarded, except, if due to the repeated removal of other assignments, a_i eventually becomes the left or right bound of the current domain. Thus, bound-consistency is a weaker concept than domain-consistency.

Example 16.2.9

We again examine the Examples 16.2.4 and 16.2.7. Establishing *n-b*-consistency must lead to the domain intervals $\delta(x_1) = [4,10]$, $\delta(x_2) = [1,7]$ and $\delta(x_3) = [1,10]$. Here, the size of the reduced search space is $|\,\delta(x_1) \times \delta(x_2) \times \delta(x_3)| = 7 \cdot 7 \cdot 10 = 490$ compared with the size of the original search space (1000) and the size of the *n*-d-consistent search space (64). □

Unfortunately, the following complexity result applies.

Theorem 16.2.10

Establishing n-b-consistency for the CSP is an NP-hard problem.

Proof. Consider an instance I of the CSP. Let $\Delta = \{ \delta(x_i) \mid x_i \in \mathcal{V} \}$ be the corresponding set of current domains, such that Δ is N-b-consistent. Obviously, $\mathcal{F}(I)$ is not empty iff there exists $x_i \in \mathcal{V}$ satisfying $\delta(x_i) \neq \emptyset$. □

A similar proof shows that establishing n-d-consistency is NP-hard as well.

Consistency Tests

In general, establishing k-consistency is ruled out due to the complex data structures that are necessary for the administration of the k-feasible subsets. In the last subsection we have further seen that establishing n-d- or n-b-consistency is an NP-hard problem. Consequently, using constraint propagation in order to solve the CSP is only sensible if we content ourselves with approximations of the concepts of consistency that have been introduced.

An important problem is to derive simple rules which will lead to efficient search space reductions, but at the same time can be implemented efficiently with a low polynomial time complexity. These rules are known as *consistency tests* and are generally described through a condition-instruction pair $\mathcal{Z}$ and $\mathcal{B}$. Intuitively, the semantics of a consistency test is as follows: whenever condition $\mathcal{Z}$ is satisfied, $\mathcal{B}$ has to be executed. $\mathcal{Z}$ may be, for instance, an equation or inequality, while $\mathcal{B}$ may be a domain reduction rule. We will often use the shorthand notation $\mathcal{Z} \Rightarrow \mathcal{B}$ for consistency tests.

Example 16.2.11

Let us derive a consistency test for the CSP instance I described in Example 16.2.3. Consider the constraint (*vi*) $x_2 + x_3 \geq 6$. Given an assignment a_2 of x_2, we can remove a_2 from $\delta(x_2)$ if there exists no assignment $a_3 = \delta(x_3)$ satisfying (*vi*). However, we do not really have to test all assignments in $\delta(x_3)$, because if (*vi*) is not satisfied for $a_3 = \max \delta(x_3)$ then it is not satisfied for any other assignment in $\delta(x_3)$ and vice versa. Hence, for any $a_2 \in \mathcal{D}(x_2)$,

$$\gamma(a_2) : a_2 + \max \delta(x_3) < 6 \;\Rightarrow\; \delta(x_2) := \delta(x_2) \setminus \{a_2\}$$

defines a consistency test for I. □

Of course, this example is quite simple and it may not seem clear whether any advantages can be drawn from such elementary deductions. Surprisingly, however, an analogously simple analysis will allow us to derive powerful consistency tests for particular classes of constraints as will be seen in one of the subsequent sections.

One of our objectives is to compare consistency tests. This requires a condition which enables us to determine whether certain consistency tests are "at least as good" as certain others. Intuitively, this applies if the deductions implied by a set of consistency tests are "at least as good" as those implied by another set. In order to elaborate this rather vague description, we will focus on *domain consistency tests*, i.e. consistency tests which deduce domain reductions. Similar results, however, apply for other types of consistency tests.

Let us derive a formal definition of domain consistency tests. Let $\Theta := 2^{\mathcal{D}(x_1)} \times \cdots \times 2^{\mathcal{D}(x_n)}$, where $2^{\mathcal{D}(x_i)}$ denotes the set of all subsets of $\mathcal{D}(x_i)$. Given $\Delta, \Delta' \in \Theta$, that is, $\Delta = \{ \delta(x_i) \mid x_i \in \mathcal{V} \}$ and $\Delta' = \{ \delta'(x_i) \mid x_i \in \mathcal{V} \}$, we say that

1. $\Delta \subseteq \Delta'$ iff $\delta(x_i) \subseteq \delta'(x_i)$ for all $x_i \in \mathcal{V}$,
2. $\Delta \subsetneq \Delta'$ iff $\Delta \subseteq \Delta'$, and there exists $x_i \in \mathcal{V}$, such that $\delta(x_i) \subsetneq \delta'(x_i)$.

Domain consistency tests have to satisfy two conditions. First, current domains are either reduced or left unchanged. Second, only assignments $a_i \in \delta(x_i)$ are removed for which no feasible assignment $a = (a_1, \cdots, a_i, \cdots, a_n)$ exists, because otherwise solutions would be lost. Since, however, we do not need the second condition in order to derive the results of this section, only the first one is formalized.

Definition 16.2.12

A domain consistency test γ is a function $\gamma : \Theta \to \Theta$ satisfying $\gamma(\Delta) \subseteq \Delta$ for all $\Delta \in \Theta$.

Suppose now that a set of domain consistency tests is given. In order to obtain the maximal domain reduction possible, these tests have to be applied repeatedly in an iterative fashion rather than only once. The reason for this is that, after the reduction of some domains, additional domain adjustments can possibly be derived using some of the tests which have previously failed in deducing any reductions. This has been demonstrated, for instance, in Example 16.2.3. Thus, the deduction process should be carried out until no more adjustments are possible or, in other words, until the set Δ of current domains becomes a fixed point. The standard fixed point procedure is shown in Algorithm 16.2.13.

Algorithm 16.2.13 *Fixed point*

Input: Δ: set of current domains;

```
begin
  repeat
    Δ_old := Δ;
    for all (γ ∈ Γ) do Δ := γ(Δ);  -- Γ is a set of consistency tests
  until (Δ := Δ_old);
end;
```

It is important to mention that the fixed point computed does not have to be unique and usually depends upon the order of the application of the consistency tests. For this reason we will only study *monotonous* consistency tests for which the order of application does not affect the outcome of the domain reduction process. This result will be derived in the following.

Definition 16.2.14

A consistency test γ is *monotonous* iff the following condition is satisfied:

$$\forall \Delta, \Delta' \in \Theta : \Delta \subseteq \Delta' \Rightarrow \gamma(\Delta) \subseteq \gamma(\Delta') . \tag{16.2.1}$$

Let us first define the Δ-fixed-point mentioned above. Let Γ be a set of monotonous domain consistency tests. For practical reasons we will always assume that Γ is finite. Let $\gamma_\infty := (\gamma_g)_{g \in I\!N} \in \Gamma^{I\!N}$ be a series of domain consistency tests in Γ, such that

$$\forall\, \gamma \in \Gamma, \forall\, h \in I\!N, \exists\, g > h : \gamma_g = \gamma . \tag{16.2.2}$$

The series γ_∞ determines the order of application of the consistency tests. The last condition ensures that every consistency test in Γ is (a priori) infinitely often applied. Starting with an arbitrary set Δ of current domains, we define the series of current domain sets $(\Delta_g)_{g \in I\!N}$ induced by γ_∞ through the following recursive equation

$$\Delta_0 := \Delta ,$$
$$\Delta_g := \gamma_g(\Delta_{g-1}) .$$

Since all domains $\mathcal{D}(x_i)$ are finite and $\Delta_g \subseteq \Delta_{g-1}$ due to Definition 16.2.12, there obviously exists $g^* \in I\!N$, such that $\Delta_g = \Delta_{g^*}$ for all $g \geq g^*$. We can therefore define $\gamma_\infty(\Delta) := \Delta_{g^*}$. The next question to answer is whether $\gamma_\infty(\Delta)$ really depends on the chosen series γ_∞.

Theorem 16.2.15 *Unique fixed points*. [DPP00].

If Γ *is a set of monotonous domain consistency tests and* $\gamma_\infty , \gamma'_\infty \in \Gamma^{I\!N}$ *are series satisfying* (16.2.2) *then* $\gamma_\infty(\Delta) = \gamma'_\infty(\Delta)$.

Proof. For reasons of symmetry we only have to show $\gamma_\infty(\Delta) \subseteq \gamma'_\infty(\Delta)$.
Let $(\Delta_g)_{g \in I\!N}$ and $(\Delta'_{g'})_{g' \in I\!N}$ be the series induced by γ_∞ and γ'_∞ respectively. It is sufficient to prove that for all $g' \in I\!N$, there exists $g \in I\!N$, such that $\Delta_g \subseteq \Delta'_{g'}$. This simple proof will be carried out by induction.

The assertion is obviously true for $g' = 0$. For $g' > 0$, we have $\Delta'_{g'} = \gamma'_{g'}(\Delta'_{g'-1})$. By the induction hypothesis, there exists $h \in I\!N$, such that $\Delta_h \subseteq \Delta'_{g'-1}$. Further, (16.2.2) implies that there exists $g > h$ satisfying $\gamma_g = \gamma'_{g'}$. Since $g > h$, we know

that $\Delta_{g-1} \subseteq \Delta_h$. Using the monotony property of γ_g, we can conclude

$$\Delta_g = \gamma_g(\Delta_{g-1}) \subseteq \gamma_g(\Delta_h) \subseteq \gamma_g(\Delta'_{g'-1}) = \gamma'_{g'}(\Delta'_{g'-1}) = \Delta'_{g'}.$$

This completes the induction proof. □

Definition 16.2.16

Let Γ be a set of monotonous domain consistency tests, Δ a set of current domains and $\gamma_\infty \in \Gamma^{I\!N}$ an arbitrary series satisfying (16.2.2). We define $\Gamma(\Delta) := \gamma_\infty(\Delta)$ to be the unique Δ-*fixed-point* induced by Γ and Δ.

Based on these observations, we can now propose a dominance criterion for domain consistency tests.

Definition 16.2.17

Let Γ, Γ' be sets of monotonous consistency tests.

1. Γ *dominates* Γ' ($\Gamma \trianglerighteq \Gamma'$) iff $\Gamma(\Delta) \subseteq \Gamma'(\Delta)$ for all $\Delta \in \Theta$.
2. Γ *strictly dominates* Γ' ($\Gamma \succ \Gamma'$) iff $\Gamma \trianglerighteq \Gamma'$, and there exists $\Delta \in \Theta$, such that $\Gamma(\Delta) \subsetneq \Gamma'(\Delta)$.
3. Γ is *equivalent* to Γ' ($\Gamma \sim \Gamma'$) iff ($\Gamma \trianglerighteq \Gamma'$) and ($\Gamma' \trianglerighteq \Gamma$).

The next theorem provides a simple condition for testing dominance of domain consistency tests. Basically, the theorem states that a set of domain consistency tests Γ dominates another set Γ' if all domain reductions implied by the tests in Γ' can be simulated by a finite number of tests in Γ.

Theorem 16.2.18

Let Γ, Γ' *be sets of monotonous consistency tests. If for all* $\gamma' \in \Gamma'$ *and all* $\Delta \in \Theta$, *there exist* $\gamma^1, \cdots, \gamma^d \in \Gamma$, *so that*

$$(\gamma^d \circ \ldots \circ \gamma^1)(\Delta) \subseteq \gamma'(\Delta) \tag{16.2.3}$$

then $\Gamma \trianglerighteq \Gamma'$.

Proof. Let γ_∞ and $\gamma'_\infty \in \Gamma^{I\!N}$ be series satisfying (16.2.2). Let, further, $(\Delta_g)_{g \in I\!N}$ and $(\Delta'_{g'})_{g' \in I\!N}$ be the series induced by γ_∞ and γ'_∞ respectively. Again, we will prove by induction that for all $g' \in I\!N$, there exists $g \in I\!N$, such that $\Delta_g \subseteq \Delta'_{g'}$, since this immediately implies $\Gamma(\Delta) \subseteq \Gamma'(\Delta)$.

The assertion is obviously true for $g' = 0$. Therefore, let $g' > 0$ and $\Delta'_{g'} = \gamma'_{g'}(\Delta'_{g'-1})$. By the induction hypothesis, there exists $h \in I\!N$, such that $\Delta_h \subseteq \Delta'_{g'-1}$.

Let $\gamma^1, \cdots, \gamma^d \in \Gamma$ be the sequence of consistency tests satisfying (16.2.3) for

$\gamma'_{g'}$ and Δ_h. There exist $g_d > \ldots > g_1 > h$ satisfying $\gamma_{g_1} = \gamma^1, \ldots, \gamma_{g_d} = \gamma^d$ due to (16.2.2). Without loss of generality, we assume that $g_d = h+d, \ldots, g_1 = h+1$, so that

$$\Delta_{h+d} = (\gamma_{h+d} \circ \ldots \circ \gamma_{h+1})(\Delta_h) \subseteq \gamma'_{g'}(\Delta_h) \subseteq \gamma'_{g'}(\Delta'_{g'-1}) = \Delta'_{g'}$$

which proves the induction step. This verifies the dominance relation $\Gamma \trianglerighteqq \Gamma'$. □

Example 16.2.19

Let us reconsider the consistency tests derived in Example 16.2.11:

$$\gamma(a_2) : a_2 + \max \delta(x_3) < 6 \;\Rightarrow\; \delta(x_2) := \delta(x_2) \setminus \{a_2\} \; .$$

Instead of defining a consistency test for each $a_2 \in \mathcal{D}(x_2)$, it is sufficient to apply a single consistency test to obtain the same effects. Observe that if a_2 can be removed then all assignments $a'_2 < a_2$ can be removed as well, so that we can replace $a_2 \in \delta(x_2)$ with $\min \delta(x_2)$. This leads to the consistency test:

$$\gamma : \min \delta(x_2) + \max \delta(x_3) < 6 \;\Rightarrow\; \delta(x_2) := \delta(x_2) \setminus \{ \min \delta(x_2) \} \; .$$

Obviously, if a_2 can be removed from $\delta(x_2)$ using $\gamma(a_2)$ then γ removes a_2 after at most $a_2 - \min \delta(x_2) + 1$ steps. Thus, $\Gamma := \{\gamma\}$ dominates $\Gamma' := \{ \gamma(a_2) \mid a_2 \in \mathcal{D}(x_2) \}$. Accordingly, Γ' dominates Γ, because $\Gamma' \supseteq \Gamma$. This proves that Γ and Γ' are equivalent. □

16.3 The Disjunctive Scheduling Problem

The *disjunctive scheduling problem* (*DSP*) is a natural generalization of important scheduling problems like the job shop scheduling problem (JSP) which has been extensively studied in the last decades, or the open shop scheduling problem (OSP) which only in recent years has attracted more attention in scheduling research.

The *DSP* can be described as follows [Pha00]: a finite set of tasks each of which has a specific processing time, has to be scheduled with the objective of minimizing the *makespan*, i.e. the maximum of the completion times of all tasks. Preemption is not allowed which means that tasks must not be interrupted during their processing. In general, tasks cannot be processed independently from each other due to additional technological requirements or scarcity of resources. The DSP considers two kinds of constraints between pairs of tasks which model special classes of restrictions: *precedence* and *disjunctive constraints*.

- *Precedence constraints* which are also known as *temporal constraints* specify a fixed processing order between pairs of tasks. Precedence constraints cover technological requirements of the kind that some task T_i must finish before

another task T_j can start, for instance, if the output of T_i is the input of T_j.

- *Disjunctive constraints* prevent the simultaneous or overlapping processing of tasks without, however, specifying the processing order. If a disjunctive constraint is defined between two tasks T_i and T_j then one of the alternatives "T_i before T_j" or "T_j before T_i" must be enforced, but which one is not predetermined. Disjunctive constraints model the resource demand of tasks in a scheduling environment with scarce resource supply. More precisely, the capacity of each resource like special machines, tools or working space is one unit per period of processing time. Tasks use at most a (constant) unit amount of each resource per processing period. Due to the limited amount of resources, two tasks requiring the same resource cannot be processed in parallel.

Note that the term disjunctive constraint, as introduced here and as commonly used in scheduling, is a special case of the general concept of disjunctive constraints.

The DSP and its subclasses have been extensively studied in academic research, since its simple formulation, on the one hand, and its intractability, on the other hand, make it a perfect candidate for the development and analysis of efficient solution techniques. Indeed, the solution techniques that have been derived for the DSP have contributed a lot to the improvement of methods for less idealized and more practice oriented problems. Extensions of the DSP generally consider sequence-dependent setup times, minimal and maximal time lags, multipurpose and parallel machines, non-unit resource supply and demand, machine breakdowns, stochastic processing times, etc.

Section 16.3.1 formulates the DSP as a constraint optimization problem with disjunctive constraints as proposed by Roy and Sussman [RS64] for the JSP. The strength of this model becomes apparent later once the common graph theoretical interpretation of the disjunctive scheduling model is presented. In Section 16.3.2, solution methods for the DSP that are based on constraint propagation are briefly discussed.

16.3.1 The Disjunctive Model

Let $B = \{1, \cdots, n\}$ be the index set of tasks to be scheduled. The processing time of task T_i, $i \in B$ is denoted with p_i. By choosing sufficiently small time units, we can always assume that the processing times are positive integer values. With each task there is associated a start time domain variable st_i with domain set $\mathcal{D}(st_i) = \mathbb{N}_0$.

If a precedence or disjunctive constraint is defined between two tasks then we say that these tasks are in *conjunction* or *disjunction* respectively. The tasks in conjunction are specified by a relation $C \subseteq B \times B$. If $(i,j) \in C$ then task T_i has to finish before task T_j can start. Instead of writing $(i,j) \in C$ we will therefore use the more suggestive $i \to j \in C$. The tasks in disjunction are specified by a

symmetric relation $D \subseteq B \times B$. Whenever $(i,j) \in D$, tasks T_i and T_j cannot be processed in parallel. Since $(i,j) \in D$ implies $(j,i) \in D$, we will write $i \leftrightarrow j \in D$. Finally, let $Z = \{ p_i \mid i \in B \}$ be the set of processing times.

An instance of the DSP is uniquely determined by the tuple $I = (B, C, D, Z)$. Since we want to minimize the makespan, i.e. the maximal completion time of all tasks, the objective function is $C_{max}(I) = \max_{i \in B}\{st_i + p_i\}$. The DSP can be written as follows:

$$
\begin{array}{lll}
\textit{minimize } \{C_{max}(I)\} & & \\
st_i \in \mathcal{D}(st_i) = I\!N_0 & & i \in B, \\
(i) \quad st_i + p_i \le st_j & & i \to j \in C, \\
(ii) \quad st_i + p_i \le st_j \quad \vee \quad st_j + p_i \le st_i & & i \leftrightarrow j \in D.
\end{array}
$$

Let us first define an assignment $ST = (st_1, \cdots, st_n) \in \mathcal{D}(st_1) \times \cdots \times \mathcal{D}(st_n)$ of all start time variables. For the sake of simplicity, we will use the same notation for variables and their assignments. An assignment *ST* is *feasible*, i.e. it defines a schedule (cf. Section 3.1), if it satisfies all precedence constraints (*i*) and all disjunctive constraints (*ii*). Reformulating the DSP, the problem is to find a feasible schedule with minimal objective function value $C_{max}(I)$. Obviously, for each instance of the DSP, there exists a feasible and optimal schedule.

A Graph Theoretical Approach

The significance of the disjunctive scheduling model for the development of efficient solution methods is revealed if we consider its graph theoretical interpretation. In analogy to Section 10.1, a *disjunctive graph* is a weighted graph $\mathcal{G} = (B, C, D, W)$ with node set *B*, arc sets *C*, $D \subseteq B \times B$ where *D* is symmetric, and weight set *W*. *C* is called the set of precedence arcs, *D* the set of disjunctive arcs. Each arc $i \to j \in C \cup D$ is labelled with a weight $w_{i \to j} \in W$. Since *D* is symmetric, we will represent disjunctive arcs as doubly directed arcs and sometimes refer to $i \leftrightarrow j$ as a disjunctive edge. Notice that $i \leftrightarrow j \in D$ is labelled with two possibly different weights, $w_{i \to j}$ and $w_{j \to i}$.

Let $I = (B, C, D, Z)$ be an instance of the *DSP*. In order to define the associated disjunctive graph $\mathcal{G}(I)$, we first introduce two dummy tasks *start* (0) and *end* $(*)$ so as to obtain a connected graph. Obviously, *start* precedes all tasks, while *end* succeeds all tasks. Further, the processing times of *start* and *end* are zero.

Definition 16.3.1

If $I = (B, C, D, Z)$ is an instance of the DSP then $\mathcal{G}(I) := (B^*, C^*, D, W)$ is the associated disjunctive graph, where

$$B^* := B \cup \{0, *\},$$

$$C^* := C \cup \{ 0 \to i \mid i \in B \cup \{*\} \} \cup \{ i \to * \mid i \in B \cup \{0\} \},$$
$$W = \{ w_{i \to j} = p_i \mid i \to j \in C^* \cup D \} .$$

Example 16.3.2

Let $I = (B, C, D, Z)$ be an instance of the *DSP* with $B = \{1, \cdots, 8\}$, $C = \{ 1 \to 2 \to 3, 4 \to 5, 6 \to 7 \to 8 \}$ and $D = \{ 1 \leftrightarrow 4, 1 \leftrightarrow 6, 4 \leftrightarrow 6, 2 \leftrightarrow 7, 3 \leftrightarrow 5, 3 \leftrightarrow 8, 5 \leftrightarrow 8 \}$. The corresponding disjunctive graph $\mathcal{G} = (B^*, C^*, D, W)$ is shown in Figure 16.3.1.[2] □

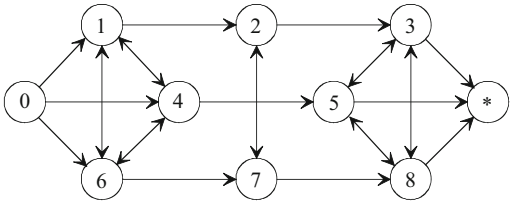


Figure 16.3.1 *A disjunctive graph.*

A disjunctive graph is transformed into a directed graph by orienting disjunctive edges.

Definition 16.3.3

Let $\mathcal{G} = (B, C, D, W)$ be a disjunctive graph, and $S \subseteq D$.

1. S is a *partial selection* iff $i \to j \in S$ implies $j \to i \notin S$ for all $i \leftrightarrow j \in D$.
2. S is a *complete selection* iff either $i \to j \in S$ or $j \to i \in S$ for all $i \leftrightarrow j \in D$.
3. A complete selection S is *acyclic* iff the directed graph $\mathcal{G}_S = (B, C \cup S)$ is acyclic.

Thus, we obtain a complete (partial) selection if (at most) one edge orientation is chosen from each disjunctive edge $i \leftrightarrow j \in D$. The selection is acyclic if the resulting directed graph is acyclic, ignoring any remaining undirected disjunctive edges. There is a close relationship between complete selections and schedules (let us remind that schedules are always feasible, as defined in Section 3.1). Indeed, if we are only interested in optimal schedules, then it is sufficient to search through the space of all selections which is of cardinality $2^{|D|}$ instead of the space of all schedules which is of cardinality $|IN_0|^n$. The DSP can thus be restated as a graph theoretical problem: find a complete and acyclic selection, such that the length of the longest path in the associated directed graph is minimal.

[2] We have not depicted all of the trivial edges involving the dummy operations *start* and *end*. Further, the specification of the weights has been omitted.

16.3.2 Solution Methods for the DSP

Countless is the number of solution methods proposed for the JSP which constitutes the most famous subclass of the DSP. A detailed survey is provided by Błażewicz et al. in [BDP96]. We only focus on solution methods which have incorporated constraint propagation techniques in some way or another. Particularly, constraint propagation has been used in exact solution methods most of which are based on a search space decomposition approach of the branch-and-bound kind. It seems fair to say that the advances in solving the JSP that have been made in the last decade can be attributed to a large extent to the development of efficient constraint propagation techniques. Undoubtedly, the algorithm of Carlier and Pinson presented in [CP89] marked a milestone in the JSP history, since for the first time an optimal solution for the notorious 10×10 problem instance proposed by Muth and Thompson [MT63] has been found and its optimality proven. Amazingly, due to the evolution of solution techniques and growing computational power, this formerly unsolvable instance can now be solved within several seconds. Important contributions towards this state of the art have been made among others by Applegate and Cook [AC91], Carlier and Pinson [CP90], Brucker et al. [BJS94, BJK94], Caseau and Laburthe [CL95], Baptiste and Le Pape [BL95] and Martin and Shmoys [MS96], to name only a few. In addition to using constraint propagation techniques in exact solution methods, the opinion eventually gains ground that combining constraint propagation with heuristic solution methods is most promising. Advances in this direction have been reported by Nuijten [Nui94], Pesch and Tetzlaff [PT96], Phan Huy [Pha96] and Nuijten and Le Pape [NL98].

16.4 Constraint Propagation and the DSP

In Section 16.2.2, constraint propagation has been introduced as an elementary method of search space reduction for the CSP or the COP. In this section, we examine how constraint propagation techniques can be adapted to the DSP. An important issue is the computational complexity of the techniques applied which has to be weighed against the search space reduction obtained. Recall that establishing *n*-, *n-d*- and *n-b*-consistency for instances of the CSP or the COP are NP-hard problems. It is not difficult to show that the same complexity result applies if we confine ourselves to the more special DSP. Thus, if constraint propagation is to be of any use in solving the DSP, we will have to content ourselves with approximations of the consistency levels mentioned above.

In the past years, two constraint propagation approaches have been studied with respect to the DSP: a time oriented and a sequence oriented approach. The time oriented approach is based on the concept of domain or bound-consistency. Each task has a current domain of possible start times. *Domain consistency tests* remove inconsistent start time assignments from current domains and, by this,

reduce the set of schedules that have to be examined. In contrast to the time oriented approach, the sequence oriented approach reduces the set of complete selections by detecting sequences of tasks, i.e. selecting disjunctive edge orientations which must occur in every optimal solution. Hence, the latter approach has been often labelled *immediate selection* (see e.g. [CP89, BJK94]) or *edge-finding* (see e.g. [AC91]). We will use the term *sequence consistency test* as used in [DPP99].

Domain and sequence consistency tests are two different concepts which complement each other. Often, a situation occurs in which either only reductions of the current domains or only edge orientations are deducible. The best results, in fact, are obtained by applying both types of consistency tests, as fixing disjunctive edges may initiate additional domain reductions and vice versa.

Section 16.4.1 introduces some notation which will be used later. The subsequent sections are concerned with the definition of domain and sequence consistency tests for the *DSP*. For the sake of simplicity, precedence and disjunctive constraints will be treated separately. At first, the simple question of how to implement constraint propagation techniques for precedence constraints is discussed in Sections 16.4.2.

In Sections 16.4.3 through 16.4.8, disjunctive constraints are examined, and both already known and new consistency tests will be presented. We assume that precedence constraints are not defined and that all tasks are in disjunction which leads to the special case of a single-machine scheduling problem [Car82].

Section 16.4.3 examines which consistency tests have to be applied in order to establish *lower-level bound-consistency*, that is, strong 3-*b*-*consistency*. Sections 16.4.4 and 16.4.5 present the well-known *input/output* and *input/output negation* consistency tests first proposed by Carlier and Pinson [CP89] and compare different time bound adjustments. Section 16.4.6 describes a class of new consistency tests which is based on the *input-or-output* conditions and is due to Dorndorf et al. [DPP99]. Section 16.4.7 takes a closer look at the concept of *energetic reasoning* proposed by Erschler et al. [ELT91] and classifies this concept with respect to the other consistency tests defined. Section 16.4.8, finally, deals with a class of consistency tests known as *shaving* which has been introduced by Carlier and Pinson [CP94] and Martin and Shmoys [MS96].

In Section 16.4.9, the results for the disjunctive constraints are summarized. Finally, Section 16.4.10 discusses how to interleave the application of the precedence and disjunctive consistency tests derived. It is worthwhile to mention that a separate analysis of precedence and disjunctive constraints leads to weaker consistency tests as compared to cases where both classes of constraints are *simultaneously* evaluated. However, it remains an open question whether simple and efficient consistency tests can be developed in this case.

16.4.1 Some Basic Definitions

For the rest of this subsection, let $I = (B, C, D, Z)$ be an instance of the DSP. Each

task T_i, $i \in B$ has a current domain $\delta(st_i) \subseteq \mathcal{D}(st_i)$. In order to avoid misinterpretations between the start time variable st_i and its assignment (for which the notation st_i is used as well), we will write δ_i instead of $\delta(st_i)$. We assume that some real or hypothetical upper bound UB on the optimal makespan is known or given, so that actually $\delta_i \subseteq [0, UB - p_i]$. This is necessary, since most of the consistency tests derived only deduce domain reductions or edge orientations if the current domains are finite. In general, the tighter the upper bound, the more information can be derived.

The earliest and latest start time of task T_i are given by $est_i := \min \delta_i$ and $lst_i := \max \delta_i$. We will interpret δ_i as an *interval* of start times, i.e. $\delta_i = [est_i, lst_i] = \{ est_i, est_i + 1, \cdots, lst_i - 1, lst_i \}$, although a set oriented interpretation is possible as well. We also need the earliest and latest completion time $ect_i := est_i + p_i$ and $lct_i := lst_i + p_i$ of task T_i.

Sometimes, it is important to distinguish between the earliest and latest start time *before* and *after* a domain reduction. We will then use the notation est_i^* and lst_i^* for the adjusted earliest and latest start times. We will often examine subsets $A \subseteq B$ of tasks and define $p(A) := \sum_{i \in A} p_i$, $EST_{min}(A) := \min_{i \in A} est_i$, and $LCT_{max}(A) := \max_{i \in A} lct_i$. Finally, $C_{max}(p_\delta(A))$ and $C_{max}(p_\delta^{pr}(A))$ denote the optimal makespan if all tasks in A are scheduled within their current domains without preemption or with preemption allowed.

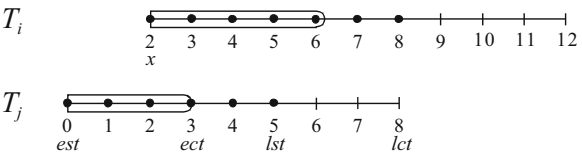


Figure 16.4.1 *Two tasks T_i, T_j with $p_i = 4$ and $p_j = 3$.*

Examples of consistency tests will be illustrated as in Figure 16.4.1 [Nui94] which shows two tasks T_i and T_j. For task T_j, the interval $[est_j, lct_j] = [0,8]$ of times at which T_j may be in process is shown as a horizontal line segment. Possible start times $[est_j, lst_j] = [0,5]$ are depicted as black circles, while the remaining times $[lst_j+1, lct_j] = [6,8]$ are marked with tick marks. A piston shaped bar of size $p_j = 3$, starting at $est_j = 0$, indicates the processing time of task T_j. The chosen representation is especially well-suited for describing the effect of domain consistency tests. If a starting time is proven to be inconsistent then the corresponding time will be marked with an x, as for instance the start time 2 on the time scale of task T_i.

16.4.2 Precedence Consistency Tests

Precedence constraints determine the order in which two specific tasks T_i and T_j have to be processed. If, for instance, task T_i has to finish before task T_j can start, then the earliest start time of T_j has to be greater than or equal to the earliest completion time of T_i. Likewise, an upper bound of the latest completion time of T_i is the latest start time of T_j. This proves the following well-known theorem.

Theorem 16.4.1 *Precedence consistency test.*

If $i, j \in B$ *and* $i \to j \in C$ *then the following domain reduction rules apply:*

$$est_j := \max\{ est_j, est_i + p_i \}, \tag{16.4.1}$$

$$lst_i := \min\{ lst_i, lst_j - p_i \}. \tag{16.4.2}$$

Of course, applying the consistency tests (16.4.1) and (16.4.2) until no more updates are possible is equivalent to the computation of a longest (precedence) path in the disjunctive graph, see [Chr75] for a standard algorithm. This algorithm traverses all tasks in a topological order which ensures that (16.4.1) and (16.4.2) only have to be applied *once* for each precedence arc.

16.4.3 Lower-Level Bound-Consistency

From this Section through Section 16.4.8, we will study the more interesting class of disjunctive constraints. For the sake of simplicity, we assume that B is a clique, i.e. all tasks in B are in disjunctions. We, further, assume that the set of precedence constraints is empty. We will, at first, discuss how disjunctive constraints interact with respect to some concept of consistency. For two reasons we opted for bound-consistency as the concept of consistency to work with. First of all, bound-consistency requires the least amount of storage capacity, since the current domains can be interpreted as intervals, so only the earliest and latest start times have to be memorized. Second, the most powerful consistency tests described in the following only affect/use the earliest and latest start times. Indeed, no efficient consistency tests which make use of "inner" start times are currently known.

Symbol	Description
$\gamma_{A,i}^{(h)}$	$h \le 4$: output consistency test for the couple (A, i), $h \ge 5$: input negation consistency test for the couple (A, i)
δ_i	current domain of T_i : $\delta_i \subseteq I\!N_0$
est_i	earliest start time of T_i : $est_i = \min\ \delta_i$
est_i^*	adjusted earliest start time of T_i
ect_i	earliest completion time of T_i : $ect_i = est_i + p_i$
lct_i	latest completion time of T_i : $lct_i = lst_i + p_i$
lst_i	latest start time of T_i : $lst_i = \max\ \delta_i$
lst_i^*	adjusted latest start time of T_i
$p_i(t_1, t_2)$	interval processing time of T_i in the time interval $[t_1, t_2)$
$[t_1, t_2)$	time interval: $[t_1, t_2) = \{\ t_1, t_1 + 1, \cdots, t_2 - 1\ \}$
$[t_1, t_2]$	time interval: $[t_1, t_2] = \{\ t_1, t_1 + 1, \cdots, t_2\ \}$
A	subset of tasks: $A \subseteq B$
$A \to i\ \ (i \to A)$	T_i has to be processed after (before) all tasks in A
$C_{max}(p_\delta(A))$	optimal makespan if all tasks in A are scheduled without preemption
$C_{max}(p_\delta^{pr}(A))$	optimal makespan if all tasks in A are scheduled with preemption allowed
$\Gamma_{\neg in}(h)$	set of input negation consistency tests
$\Gamma_{out}(h)$	set of output consistency tests
$EST_{min}(A)$	minimal earliest start time in A : $EST_{min}(A) = \min_{i \in A}\{est_i\}$
$LB_h(A)$	time bound adjustment for output consistency tests
$LB_h(A, i)$	time bound adjustment for input negation consistency tests
$LCT_{max}(A)$	maximal latest completion time in A : $LCT_{max}(A) = \max_{i \in A}\{lct_i\}$
$B_{(t_1,t_2)}$	subset of tasks which must be processed completely or partially in the time interval $[t_1, t_2)$: $B_{(t_1,t_2)} = \{\ i \in B \mid p_i(t_1, t_2) > 0\ \}$
$p(A)$	sum of processing times in A : $p(A) = \sum_{i \in A} p_i$
$p(A, t_1, t_2)$	sum of interval processing times in A in the time interval $[t_1, t_2)$: $p(A, t_1, t_2) = \sum_{i \in A} p_i(t_1, t_2)$
$\mathcal{T}(A)$	task set of A : $\mathcal{T}(A) = \mathcal{T}(EST_{min}(A), LCT_{max}(A))$
$\mathcal{T}(t_1, t_2)$	task set: $\mathcal{T}(t_1, t_2) = \{\ i \in B \mid t_1 \le est_i\ ,\ lct_i \le t_2\ \}$

Table 16.4.1: *List of symbols.*

Our goal is to examine which domain consistency tests have to be applied in order to establish strong 3-*b*-consistency which is also known as *lower-level bound-consistency*. 1-*b*-consistency is trivially established, since unary con-

straints are not involved, so only 2-*b*- and 3-*b*-consistency remain to be studied.

The corresponding consistency tests will be derived through an elementary and systematic evaluation of all constraints. This “bottom up" approach is quite technical, but it closes the gap that is usually left by the consistency tests which are due to the researcher's inspiration and insight into the problem's nature. As a consequence, we will rediscover most of these consistency tests which have been “derived" in a “top down" fashion in a slightly stronger version.

2-b-Consistency

In order to test for 2-*b*-consistency, pairs of different tasks have to be examined. If T_i , $i \in B$ is a task and $st_i \in \{est_i, lst_i\}$ an assignment of its start time, then st_i is (currently) consistent and cannot be removed if there exists another task T_j, $j \in B$, and an assignment $st_j \in \delta_j$, such that st_i and st_j satisfy the disjunctive constraint $i \leftrightarrow j$:

$$\exists\, st_j \in \delta_j : st_i + p_i \le st_j \;\vee\; st_j + p_j \le st_i\,. \tag{16.4.3}$$

Of course, if (16.4.3) is satisfied for all pairs (i,j) then 2-*b*-consistency is established. Since $\delta_j = [est_j, lst_j]$, this condition can be simplified as follows:

$$st_i + p_i \le lst_j \;\vee\; est_j + p_j \le st_i\,. \tag{16.4.4}$$

Suppose now that 2-*b*-consistency is not yet established. We will first show how to derive a well-known consistency test which removes an inconsistent assignment $st_i = est_i$ through a simple evaluation of (16.4.4). Similar arguments lead to a consistency test for removing the assignment $st_i = lst_i$. These consistency tests have been first proposed by Carlier and Pinson [CP89]. Obviously, if (16.4.4) is not satisfied for $st_i = est_i$ then we can remove est_i, i.e.

$$est_i + p_i > lst_j \;\wedge\; est_j + p_j > est_i \;\Rightarrow\; est_i = est_i + 1. \tag{16.4.5}$$

Observe that after adjusting est_i, the condition $est_i + p_i > lst_j$ on the left side of (16.4.5) is still satisfied. Therefore, we can increase est_i as long as $est_j + p_j > est_i$, i.e. until $est_j + p_j \le est_i$. This leads to the improved consistency test

$$est_i + p_i > lst_j \;\Rightarrow\; est_i = \max\{est_i, est_j + p_j\}. \tag{16.4.6}$$

Analogously, testing $st_i = lst_i$ leads to the consistency test

$$est_j + p_j > lst_i \;\Rightarrow\; lst_i = \min\{lst_i, lst_j - p_i\}. \tag{16.4.7}$$

Let Γ_2 be the set of consistency tests defined by (16.4.6) and (16.4.7) for all tasks $T_i \neq T_j$. The next lemma in combination with Theorem 16.2.15 ensures that there exists a unique fixed point $\Gamma_2(\Delta)$, i.e. applying the consistency tests in Γ_2 in an arbitrary order until no more updates are possible will always result in the same set of current domains.

Lemma 16.4.2

Γ_2 *is a set of monotonous consistency tests*.

Proof. For reasons of symmetry, it is sufficient to examine the consistency tests given by (16.4.6). Let $\Delta = \{ [est_l , lst_l] \mid l \in B \}$ and $\Delta' = \{ [est'_l, lst'_l] \mid l \in B \}$. If $\Delta \subseteq \Delta'$, that is, $est'_l \leq est_l$ and $lst_l \leq lst'_l$ for all $l \in B$ then

$$\begin{aligned} est'_i + p_i > lst'_j \quad &\Rightarrow \quad est_i + p_i > lst_j \\ &\overset{(13.4.6)}{\Rightarrow} \quad est^*_i = \max\{ est_i , est_j + p_j \} \\ &\Rightarrow \quad est^*_i \geq \max\{ est'_i , est'_j + p_j \} \\ &\Rightarrow \quad est^*_i \geq est'^*_i \end{aligned}$$

As all other earliest and latest start times remain unchanged, $est'^*_l \leq est^*_l$ and $lst^*_l \leq lst'^*_l$ for all $l \in B$ which proves the monotony property. □

Altogether, the following theorem has been proven, see also [Nui94].

Theorem 16.4.3

For all $\Delta \in \Theta$, $\Gamma_2(\Delta)$ *is* 2-*b-consistent*. □

Example 16.4.4

Consider the situation that has been depicted in Figure 16.4.1. Since $est_i + p_i = 6 > 5 = lst_j$, we can adjust $est_i = \max\{est_i , est_j + p_j\} = \max\{2,3\} = 3$ according to (16.4.6). Note that the current domain of task T_j remains unchanged if (16.4.7) is applied. □

$\Gamma_2(\Delta)$ can be computed by repeatedly testing all pairs $i,j \in B$, $i \neq j$, until no more updates are possible. We will discuss other algorithms which subsume the tests for 2-*b*-consistency at a later time. As a generalization of the pair test Focacci and Nuijten [FN00] have proposed two consistency tests for shop scheduling, with sequence dependent setup times between pairs of tasks processed by the same disjunctive resource.

3-b-Consistency

In order to test for 3-*b*-consistency, triples of pairwise different tasks have to be examined. Again, let T_i, $i \in B$, be a task, and $st_i \in \{est_i, lst_i\}$. The start time st_i is (currently) consistent and cannot be removed if there exist $j, k \in B$, such that i, j, k are indices of pairwise different tasks, and there exist assignments $st_j \in \delta_j$, $st_k \in \delta_k$, such that st_i, st_j, and st_k satisfy the disjunctive constraints $i \leftrightarrow j$, $i \leftrightarrow k$,

and $j \leftrightarrow k$. Let us first consider this condition for $st_i = est_i$:

$$\exists\, st_j \in \delta_j, \exists\, st_k \in \delta_k : \quad \begin{cases} (est_i + p_i \le st_j \;\vee\; st_j + p_j \le est_i) \;\wedge \\ (est_i + p_i \le st_k \;\vee\; st_k + p_k \le est_i) \;\wedge \\ (st_j + p_j \le st_k \;\vee\; st_k + p_k \le st_j)\,. \end{cases} \tag{16.4.8}$$

Again, if (16.4.8) is satisfied for all triples (i,j,k) then 3-*b*-consistency is established. This condition is equivalent to

$$\exists\, st_j \in \delta_j, \exists\, st_k \in \delta_k : \quad \begin{cases} (est_i + p_i \le st_j \;\wedge\; st_j + p_j \le st_k) \;\vee \\ (est_i + p_i \le st_k \;\wedge\; st_k + p_k \le st_j) \;\vee \\ (st_j + p_j \le est_i \;\wedge\; est_i + p_i \le st_k) \;\vee \\ (st_k + p_k \le est_i \;\wedge\; est_i + p_i \le st_j) \;\vee \\ (st_j + p_j \le st_k \;\wedge\; st_k + p_k \le est_i) \;\vee \\ (st_k + p_k \le st_j \;\wedge\; st_j + p_j \le est_i)\,. \end{cases} \tag{16.4.9}$$

Each line of (16.4.9) represents a permutation of the tasks T_i, T_j, T_k, e.g. the first line corresponds to the sequence $i \to j \to k$. Since $\delta_j = [est_j, lst_j]$ and $\delta_k = [est_k, lst_k]$, (16.4.9) is equivalent to:

$$\exists\, st_j \in \delta_j, \exists\, st_k \in \delta_k : \quad \begin{cases} (est_i + p_i \le st_j \;\wedge\; st_j + p_j \le lst_k) \;\vee & (i) \\ (est_i + p_i \le st_k \;\wedge\; st_k + p_k \le lst_j) \;\vee & (ii) \\ (est_j + p_j \le est_i \;\wedge\; est_i + p_i \le lst_k) \;\vee & (iii) \\ (est_k + p_k \le est_i \;\wedge\; est_i + p_i \le lst_j) \;\vee & (iv) \\ (est_j + p_j \le st_k \;\wedge\; st_k + p_k \le est_i) \;\vee & (v) \\ (est_k + p_k \le st_j \;\wedge\; st_j + p_j \le est_i)\,. & (vi) \end{cases} \tag{16.4.10}$$

In analogy to the case of establishing 2-*b*-consistency, we can increase $est_i := est_i + 1$ if (16.4.10) is not satisfied. However, in spite of the previous simplifications, testing (16.4.10) still is too costly, since the expression on the right side has to be evaluated for all $st_j \in \delta_j$ and $st_k \in \delta_k$. In the following lemmas, we therefore replace the conditions (*i*), (*ii*), (*v*) and (*vi*) which either contain st_j or st_k with simpler conditions.

Lemma 16.4.5

If Δ *is* 2-b-*consistent and the conditions* (*iii*) *and* (*vi*) *are not satisfied then the following equivalence relations hold:*

$$\exists\, st_j \in \delta_j, \exists\, st_k \in \delta_k : \quad \begin{cases} (est_i + p_i \le st_j \;\wedge\; st_j + p_j \le lst_k) \;\vee & (i) \\ (est_i + p_i \le st_k \;\wedge\; st_k + p_k \le lst_j) & (ii) \end{cases} \tag{16.4.11}$$

$$\Leftrightarrow\ est_i + p_i + p_j \le lst_k \ \vee\ est_i + p_i + p_k \le lst_j \tag{16.4.12}$$

$$\Leftrightarrow\ \max\{\, lct_j - est_i\,,\, lct_k - est_i\,\} \ge\ p_i + p_j + p_k \tag{16.4.13}$$

Proof. Let us prove the first equivalence. The direction $\Rightarrow$ is obvious, so only $\Leftarrow$ has to be shown. Let (16.4.12) be satisfied. Without loss of generality, we can assume that either (a) $est_i + p_i + p_j \le lst_k$ and $est_i + p_i + p_k > lst_j$, or that (b) $lst_k \ge lst_j$ if both, $est_i + p_i + p_j \le lst_k$ and $est_i + p_i + p_k \le lst_j$. Studying the two cases $est_i + p_i \ge est_j$ and $est_i + p_i < est_j$ separately, we can show that in both cases there exists $st_j \in \delta_j$, such that condition (*i*) is satisfied.

Case 1: Let $est_i + p_i \ge est_j$. If we can prove that $est_i + p_i \le lst_j$ then choosing $st_j := est_i + p_i$ is possible, as then $st_j \in [est_j\,,\, lst_j] = \delta_j$, $est_i + p_i \le st_j$ and $st_j + p_j = est_i + p_i + p_j \le lst_k$. Thus, condition (*i*) is satisfied. In order to prove $est_i + p_i \le lst_j$, we use the assumption that condition (*iii*) is not satisfied, i.e. that $est_j + p_j > est_i$ or $est_i + p_i > lst_k$. It follows from $est_i + p_i < est_i + p_i + p_j \le lst_k$ that the second inequality cannot be satisfied, so that actually $est_j + p_j > est_i$. Thus, indeed, $est_i + p_i \le lst_j$, as we have assumed 2-*b*-consistency (see (16.4.6)).

Case 2: Let $est_i + p_i \le est_j$. If $est_j + p_j \le lst_k$, setting $st_j := est_j \in \delta_j$ again satisfies condition (*i*). We now have to show that, in fact, $est_j + p_j \le lst_k$. Again, we will use the assumption that 2-*b*-consistency is established. If $est_j + p_j > lst_k$ then (16.4.7) implies $lst_k \le lst_j - p_k$ and $lst_k < lst_j$. Further, as $est_i + p_i + p_j \le lst_k \le lst_j - p_k$ we can conclude $est_i + p_i + p_k \le lst_j$. So both inequalities of (16.4.12) are satisfied, but $lst_k < lst_j$. This is a contradiction to the assumption (b).

The second equivalence is easily proven by adding p_k and p_j, respectively, on both sides of inequalities (16.4.12) . □

Lemma 16.4.6

If Δ *is* 2-*b-consistent then the following equivalence relations hold:*

$$\exists\, st_j \in \delta_j, \exists\, st_k \in \delta_k: \quad \begin{cases} (\, est_j + p_j \le st_k \ \wedge\ st_k + p_k \le est_i\,) & \vee \quad (v) \\ (\, est_k + p_k \le st_j \ \wedge\ st_j + p_j \le est_i\,) & \quad (vi) \end{cases} \tag{16.4.14}$$

$$\Leftrightarrow \quad \begin{array}{l} est_i \ge \max\{est_j + p_j + p_k\,,\, est_k + p_k\} \ \vee \\ est_i \ge \max\{est_k + p_k + p_j\,,\, est_j + p_j\} \end{array} \tag{16.4.15}$$

$$\Leftrightarrow\ \ est_i \ge \max\big\{\min\{est_j, est_k\} + p_j + p_k\,,\, est_j + p_j\,,\, est_k + p_k\big\} \tag{16.4.16}$$

Proof. We prove the first equivalence. Again, the direction $\Rightarrow$ is obvious, so we only have to show $\Leftarrow$. Let (16.4.15) be satisfied. We assume without loss of generality that $est_j \le est_k$. This implies $\max\{\, est_k + p_k + p_j\,,\, est_j + p_j\,\} \ge est_k + p_k + p_j \ge \max\{est_j + p_j + p_k,\, est_k + p_k\}$, so that $est_i \ge \max\{\, est_j + p_j + p_k\,,\, est_k + p_k\,\}$ $(*)$.

Case 1: Let $est_j + p_j \geq est_k$. If $est_j + p_j > lst_k$ then the 2-*b*-consistency (16.4.6) implies $est_j \geq est_k + p_k$ and $est_j \geq est_k$ which is a contradiction, so that actually $est_j + p_j < lst_k$. We can set $st_k := est_j + p_j \in [est_k, lst_k] = \delta_k$, and condition (*v*) is satisfied due to (∗).

Case 2: Let $est_j + p_j < est_k$. Choosing $st_k := est_k \in \delta_k$ again satisfies condition (*v*) due to (∗). A standard proof verifies the second equivalence. □

Given that 2-*b*-consistency is established, we can therefore replace (16.4.10) with the following equivalent and much simpler condition which can be tested in constant time:

$$\begin{aligned}
&(\max\{lct_j - est_i,\ lct_k - est_i\} \geq p_i + p_j + p_k) \quad \vee && (i + ii)\\
&(est_j + p_j \leq est_i \ \wedge\ est_i + p_i \leq lst_k) \quad \vee && (iii)\\
&(est_k + p_k \leq est_i \ \wedge\ est_i + p_i \leq lst_j) \quad \vee && (iv)\\
&(est_i \geq \max\{\min\{est_j, est_k\} + p_j + p_k,\ est_j + p_j,\ est_k + p_k\}) . && (v + vi)
\end{aligned} \tag{16.4.17}$$

Resuming our previous thoughts, we can increase $est_i := est_i + 1$ if (16.4.17) is not satisfied. Observe that if $(i + ii)$ is not satisfied *before* increasing est_i then it is not satisfied *after* increasing est_i. Therefore, we can proceed as follows: first, test whether $(i + ii)$ holds. If this is not the case then increase est_i until one of the conditions (*iii*), (*iv*) or $(v + vi)$ is satisfied. Fortunately, this incremental process can be accelerated by defining appropriate time bound adjustments.

Deriving the correct time bound adjustments requires a rather lengthy and painstaking analysis which is provided in Section 16.6 (Appendix). At the moment, we will only present an intuitive development of the results which avoids the distraction of the technical details.

Two cases have to be distinguished. In the first case, increasing est_i will never satisfy conditions $(i + ii)$, (*iii*) and (*iv*). This can be interpreted as the situation in which T_i can neither be processed at the first, nor at the second position, but must be processed after T_j and T_k. We then have to increase est_i until condition $(v + vi)$ is satisfied. Notice that this is always possible by choosing est_i sufficiently large, i.e. by setting

$$est_i := \max\{est_i,\ \min\{est_j, est_k\} + p_j + p_k,\ est_j + p_j,\ est_k + p_k\} .$$

However, it is possible to show that the seemingly weaker adjustment

$$est_i := \max\{est_i,\ \min\{est_j, est_k\} + p_j + p_k\}$$

is sufficient if it is combined with the tests for establishing 2-*b*-consistency or, more precisely, if after the application of this adjustment the 2-*b*-consistency tests are again applied. This leads to the following two consistency tests:

$$\max_{u\in\{i,j,k\},\, v\in\{j,k\},\, u\neq v} \{lct_v - est_u\} < p_i + p_j + p_k \quad \Rightarrow \quad est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k\}, \tag{16.4.18}$$

$$est_i + p_i > \max\{lst_j, lst_k\} \quad \Rightarrow \quad est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k\}\ . \tag{16.4.19}$$

It is both important to establish 2-*b*-consistency prior and after the application of these consistency tests, since the application of the latter test can lead to a 2-b-inconsistent state.

A generalization of these tests will be later described under the name input/output consistency tests. Trivial though it may seem, it should nevertheless be mentioned that the consistency tests (16.4.18) and (16.4.19) are not equivalent. Furthermore, observe that if the left side of (16.4.19) is satisfied then the consistency tests for pairs of tasks (16.4.6) can be applied to both (i,j) and (i,k), but may lead to weaker domain adjustments. We will give some examples which confirm these assertions.

Example 16.4.7

Consider the example depicted in Figure 16.4.2. Since

$$\max_{u\in\{i,j,k\},\, v\in\{j,k\},\, u\neq v} \{lct_v - est_u\} = 9 < 10 = p_i + p_j + p_k\,,$$

we can adjust $est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k\} = \max\{3,7\} = 7$ according to (16.4.18). By comparison, no deductions are possible using (16.4.19), as $est_i + p_i = 6 < 7 = \max\{lst_j, lst_k\}$. □

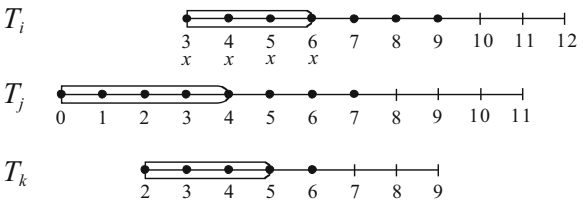


Figure 16.4.2 *Consistency test* (16.4.18).

Example 16.4.8

In Figure 16.4.3 another example is shown. Here, the consistency test (16.4.18) fails, as

$$\max_{u\in\{i,j,k\},\, v\in\{j,k\},\, u\neq v} \{lct_v - est_u\} = 9 = p_i + p_j + p_k\,.$$

The consistency test for pairs of tasks described in (16.4.6) can be applied to (i,j) and (i,k), but leaves est_j unchanged, since $est_j + p_j = est_k + p_k = 3 < 4 = est_i$. Only the consistency test (16.4.19) correctly adjusts $est_i := \max\{est_i, \min\{est_j, est_k\}$

$+ p_j + p_k\} = \max\{4,6\} = 6.$ □

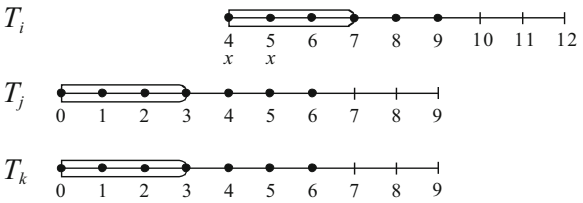


Figure 16.4.3 *Consistency test* (16.4.19).

Let us now turn to the second case in which the condition $(i + ii)$ is not satisfiable, but increasing est_i will eventually satisfy (iii) or (iv). This can be interpreted as the situation in which T_i cannot be processed first, but either $j \to i \to k$ or $k \to i \to j$ are feasible. The corresponding consistency test is as follows:

$$\begin{aligned} &\max_{v \in \{j,k\}} \{lct_v - est_i\} < p_i + p_j + p_k \\ &\qquad \Rightarrow \quad est_i := \max\{est_i, \min\{ect_j, ect_k\}\}. \end{aligned} \tag{16.4.20}$$

A generalization of this test will be later described under the name input/output negation consistency test.

Example 16.4.9

Consider the example of Figure 16.4.4. No domain reductions are possible using the consistency tests (16.4.18) and (16.4.19). Since, however, $\max_{v \in \{j,k\}}\{lct_v - est_i\} = 7 < 9 = p_i + p_j + p_k$, we can adjust $est_i := \max\{est_i, \min\{ect_j, ect_k\}\} = \max\{2, 3\} = 3$ using the consistency test (16.4.20). □

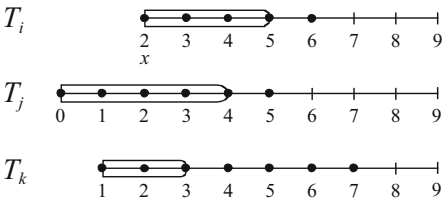


Figure 16.4.4 *Consistency test* (16.4.20).

The adjustments of the latest start times can be handled symmetrically. The same line of argumentation allows us to derive the following three consistency tests:

$$\begin{aligned} &\max_{u \in \{j,k\},\, v \in \{i,j,k\},\, u \neq v} \{lct_v - est_u\} < p_i + p_j + p_k \\ &\qquad \Rightarrow \quad lst_i := \min\{lst_i, \max\{lct_j, lct_k\} - p_j - p_k - p_i\}, \end{aligned} \tag{16.4.21}$$

$$\min\{est_j + p_j, est_k + p_k\} > lst_i \tag{16.4.22}$$

$$\Rightarrow \quad lst_i := \min\{lst_i, \max\{lct_j, lct_k\} - p_j - p_k - p_i\},$$

$$\max_{u \in \{j,k\}} \{lct_i - est_u\} < p_i + p_j + p_k \quad \Rightarrow \quad lst_i := \min\{lst_i, \max\{lst_j, lst_k\} - p_i\}. \tag{16.4.23}$$

Let Γ_3 be the set of consistency tests defined in (16.4.18)-(16.4.23) for all pairwise different triples of tasks with indices $i, j, k \in B$, and let $\Gamma_{2,3} := \Gamma_2 \cup \Gamma_3$. It can be shown that all consistency tests in $\Gamma_{2,3}$ are monotonous, so $\Gamma_{2,3}(\Delta)$ is well defined. We have proven the following theorem.

Theorem 16.4.10

For all $\Delta \in \Theta$, $\Gamma_{2,3}(\Delta)$ *is strongly* 3*-b-consistent.* □

Notice that $\Gamma_3(\Gamma_2(\Delta))$ does not have to be strongly 3-b-consistent, since the application of some of the consistency tests in Γ_3 can result in current domains which are not 2-b-consistent. So, indeed, the consistency tests in Γ_2 and Γ_3 have to be applied in alternation.

Obviously, $\Gamma_{2,3}(\Delta)$ can be computed by repeatedly testing all pairwise different pairs and triples of tasks. However, as will be seen in the following sections, there exist more efficient algorithms.

16.4.4 Input/Output Consistency Tests

In the last section, domain consistency tests for pairs and triples of tasks have been described. It suggests itself to derive domain consistency tests for a greater number of tasks through a systematic evaluation of a greater number of disjunctive constraints. For the sake of simplicity, we will refrain from this rather technical approach and follow the historical courses which finally leads to the definition of these powerful consistency tests. Note, however, that we must not expect that the consistency tests derived will establish some higher level of bound-consistency, since great store has been set on an efficient implementation.

At first, we will present generalizations of the consistency tests (16.4.18) and (16.4.19). A closer look at these tests reveals that not only domain reductions but also processing orders of tasks can be deduced. It is convenient to first introduce these sequence consistency tests so as to simplify the subsequent proofs.

Sequence Consistency Tests

Given a subset of task indices $A \subsetneq B$ and an additional task T_i, $i \notin A$, Carlier and Pinson [CP89] were the first to derive conditions which imply that T_i has to be

processed *before* or *after* all tasks T_j, $j \in A$. In the first case, they called i the *input* of A, in the second case, the *output* of A, and so the name *input/output conditions* seems justified.

Theorem 16.4.11 (*Input/Output Sequence Consistency Tests*).

Let $A \subsetneq B$ and $i \notin A$. If the input condition

$$\max_{u \in A,\, v \in A \cup \{i\},\, u \neq v} \{lct_v - est_u < p(A \cup \{i\}) \tag{16.4.24}$$

is satisfied then task T_i has to be processed before all tasks in A, for short, $i \rightarrow A$. Likewise, if the output condition

$$\max_{u \in A \cup \{i\},\, v \in A,\, u \neq v} \{lct_v - est_u\} < p(A \cup \{i\}) \tag{16.4.25}$$

is satisfied then task T_i has to be processed after all tasks in A, for short, $A \rightarrow i$.

Proof. If T_i is not processed before all tasks in A then the maximal amount of time for processing all tasks in $A \cup \{i\}$ is bounded by $\max_{u \in A,\, v \in A \cup \{i\},\, u \neq v} \{lct_v - est_u\}$. This leads to a contradiction if (16.4.24) is satisfied. Analogously, the second assertion can be shown. □

The original definition of Carlier and Pinson is slightly weaker. It replaces the input condition with

$$LCT_{max}(A \cup \{i\}) - EST_{min}(A) < p(A \cup \{i\}). \tag{16.4.26}$$

Likewise, the output condition is replaced with

$$LCT_{max}(A) - EST_{min}(A \cup \{i\}) < p(A \cup \{i\}). \tag{16.4.27}$$

We will term these conditions the *modified input/output conditions.*. There are situations in which only the input/output conditions in their stricter form lead to a domain reduction. For a discussion of the computational complexity of algorithms that implement these tests see the end of Section 16.4.

Example 16.4.12

In Example 16.4.7 (see Figure 16.4.2), we have seen that

$$\max_{u \in \{i,j,k\},\, v \in \{j,k\},\, u \neq v} \{lct_v - est_u\} = 9 < 10 = p_i + p_j + p_k,$$

so that the output (16.4.25) implies $\{j,k\} \rightarrow i$. By comparison, the modified output condition is not satisfied since

$$LCT_{max}(\{j,k\}) - EST_{min}(\{i,j,k\}) = lct_j - est_j = 11 > 10 = p_i + p_j + p_k. \quad \square$$

Domain Consistency Tests

Domain consistency tests that are based on the input/output conditions can now be simply derived. Here and later, we will only examine the adjustment of the earliest start times, since the adjustment of the latest start times can be handled analogously. Clearly, if i is the output of a subset A then T_i cannot start before all tasks of A have finished. Therefore, the earliest start time of T_i is at least $C_{max}(p_\delta(A))$, i.e. the makespan if all tasks in A are scheduled without preemption. Unfortunately, however, determining $C_{max}(p_\delta(A))$ requires the solution of the NP-hard single-machine scheduling problem [GJ79]. Thus, if the current domains are to be updated efficiently, we have to content ourselves with approximations of this bound. Some of these approximations are proposed in the next theorem which is a generalization of the consistency test (16.4.19) derived in the last subsection. This theorem is mainly due to Carlier and Pinson [CP90], Nuijten [Nui94], Caseau and Laburthe [CL95] and Martin and Shmoys [MS96]. The proof is obvious and is omitted.

Theorem 16.4.13 (*output domain consistency tests, part 1*).

If the output condition is satisfied for $A \subsetneq B$ and $i \notin A$ then the earliest start time of T_i can be adjusted to $est_i := \max\{est_i, LB_h(A)\}$, $h \in \{1,2,3,4\}$, where

(*i*) $LB_1(A) := \max_{u \in A}\{ect_u\}$,

(*ii*) $LB_2(A) := EST_{min}(A) + p(A)$,

(*iii*) $LB_3(A) := C_{max}(p_\delta^{pr}(A))$,

(*iv*) $LB_4(A) := C_{max}(p_\delta(A))$. □

Dominance Relations

Let us compare the domain reductions that are induced by the output domain consistency tests and the different bounds. For each $h \in \{1,2,3,4\}$, we denote with $\Gamma_{out}(h) := \{ \gamma_{A,i}^{(h)} \mid A \subsetneq B, i \notin A \}$ the set of output domain consistency tests defined in Theorem 16.4.13:

$$\gamma_{A,i}^{(h)} := \max_{u \in A \cup \{i\},\, v \in A,\, u \neq v} \{lct_v - est_u\} < p(A \cup \{i\}) \Rightarrow est_i := \max\{est_i, LB_h(A)\}.$$

Lemma 16.4.14

The following dominance relations hold:

1. $\Gamma_{out}(1) \trianglelefteq \Gamma_{out}(3) \trianglelefteq \Gamma_{out}(4)$,
2. $\Gamma_{out}(2) \trianglelefteq \Gamma_{out}(3) \trianglelefteq \Gamma_{out}(4)$.

Proof. As $LB_3(A) \leq LB_4(A)$, the relation $\gamma^{(4)}_{A,i}(\Delta) \subseteq \gamma^{(3)}_{A,i}(\Delta)$ holds for all $A \subsetneq B$, $i \notin A$ and $\Delta \in \Theta$. Theorem 16.2.18 then implies that $\Gamma_{out}(3) \trianglelefteq \Gamma_{out}(4)$. Further, Carlier [Car82] has shown the following identity for the preemptive bound:

$$LB_3(A) = \max_{\emptyset \neq V \subseteq A} \{EST_{min}(V) + p(V)\}. \tag{16.4.28}$$

Since the maximum expression in (16.4.28) considers all single-elemented sets and A itself, $LB_1(A) \leq LB_3(A)$ and $LB_2(A) \leq LB_3(A)$. Again, using Theorem 16.2.18, we can conclude that $\Gamma_{out}(1) \trianglelefteq \Gamma_{out}(3)$ and $\Gamma_{out}(2) \trianglelefteq \Gamma_{out}(3)$. □

Intuitively, it seems natural to assume that $\Gamma_{out}(1)$ is strictly dominated by $\Gamma_{out}(3)$, while $\Gamma_{out}(3)$ is strictly dominated by $\Gamma_{out}(4)$. Indeed, this is true. Remember that, since $\Gamma_{out}(1) \trianglelefteq \Gamma_{out}(3)$ has already been shown, we only have to find an example in which $\Gamma_{out}(3)$ leads to a stronger domain reduction than $\Gamma_{out}(1)$ in order to verify $\Gamma_{out}(1) \prec \Gamma_{out}(3)$. The same naturally holds for $\Gamma_{out}(3)$ and $\Gamma_{out}(4)$.

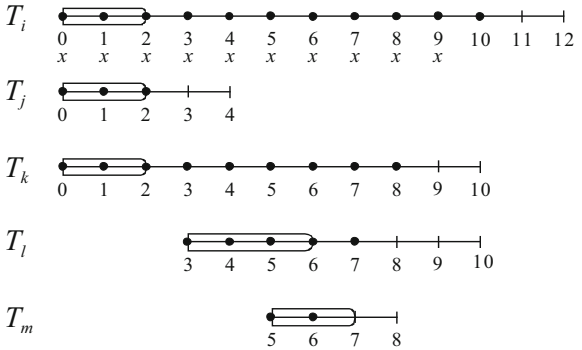
Example 16.4.15

Consider the situation illustrated in Figure 16.4.5 with five tasks with indices i, j, k, l, m. The table in Figure 16.4.5 lists all feasible sequences and the associated schedules. Examining the start times of the feasible schedules shows that the domains δ_j, δ_k, δ_l, δ_m cannot be reduced. Likewise, it can be seen that i is the output of $A = \{j,k,l,m\}$ with the earliest start time being $LB_4(A) = 10$. In fact, the output condition holds, as

$$\max_{u \in A \cup \{i\},\, v \in A,\, u \neq v} \{lct_v - est_u\} = 10 < 11 = p(A \cup \{i\}),$$

so that we can adjust est_i using one of the bounds of Theorem 16.4.13. Apart from $LB_4(A) = 10$, it is possible to show that $LB_1(A) = 7$, $LB_2(A) = 9$ and $LB_3(A) = 9$. Obviously, $LB_1(A) < LB_3(A) < LB_4(A) = 10$. Notice that, after the adjustment of est_i, no other adjustments are possible if the same lower bound is used again, so that a fixed point is reached. This confirms the conjecture $\Gamma_{out}(1) \prec \Gamma_{out}(3) \prec \Gamma_{out}(4)$. □

It remains to classify $\Gamma_{out}(2)$. Comparing $LB_1(A)$ and $LB_2(A)$ shows that all three cases $LB_1(A) < LB_2(A)$, $LB_1(A) = LB_2(A)$ and $LB_1(A) > LB_2(A)$ can occur. Further, comparing $LB_2(A)$ and $LB_3(A)$ reveals that $LB_2(A) \leq LB_3(A)$ and sometimes $LB_2(A) < LB_3(A)$. So we would presume that $\Gamma_{out}(1)$ and $\Gamma_{out}(2)$ are not comparable, while $\Gamma_{out}(2)$ is strictly dominated by $\Gamma_{out}(3)$. This time, however, our intuition fails, since in fact $\Gamma_{out}(2)$ and $\Gamma_{out}(3)$ are equivalent.



sequence	st_i	st_j	st_k	st_l	st_m
$j \to k \to m \to l \to i$	10	0	2	7	5
$j \to l \to m \to k \to i$	10	0	8	3	6
$k \to j \to m \to l \to i$	10	2	0	7	5

Figure 16.4.5 *Comparing* $\Gamma_{out}(1)$, $\Gamma_{out}(3)$ *and* $\Gamma_{out}(4)$.

Theorem 16.4.16 (*dominance relations for output consistency tests*). [DPP00]

$$\Gamma_{out}(1) \prec \Gamma_{out}(2) \sim \Gamma_{out}(3) \prec \Gamma_{out}(4).$$

Proof. We only have to prove $\Gamma_{out}(3) \trianglelefteq \Gamma_{out}(2)$. It is sufficient to show that for all $A \subsetneq B$, $i \notin A$ and all $\Delta \in \Theta$, one of the following cases applies:

(1) $\gamma^{(3)}_{A,i}(\Delta) = \gamma^{(2)}_{A,i}(\Delta)$,

(2) $\exists\, V \subsetneq A:\ \gamma^{(3)}_{A,i}(\Delta) = \gamma^{(2)}_{V,i}(\gamma^{(2)}_{A,i}(\Delta))$.

Once more, Theorem 16.2.18 will then lead to the desired result. Let us assume that the output condition (16.4.25) is satisfied for some $A \subsetneq B$ and $i \notin A$. We have to compare the bounds:

(*i*) $LB_2(A) = EST_{min}(A) + p(A)$,

(*ii*) $LB_3(A) = \max_{\emptyset \neq V \subseteq A} \{EST_{min}(V) + p(V)\}$,

If $LB_2(A) = LB_3(A)$ then $\gamma^{(2)}_{A,i}$ and $\gamma^{(3)}_{A,i}$ deduce the same domain reductions and case (1) applies. Let us therefore assume that $LB_2(A) < LB_3(A)$. Since the preemptive bound is determined by (16.4.28) , there exists $V \subset A$, $V \neq \emptyset$, such that $LB_3(A) = EST_{min}(V) + p(V)$. Since $LB_2(A) < LB_3(A)$, this is equivalent to

$$EST_{min}(A) + p(A) < EST_{min}(V) + p(V)\,. \tag{16.4.29}$$

Subtracting $p(V)$ from both sides yields

$$EST_{min}(A) + p(A - V) < EST_{min}(V) \tag{16.4.30}$$

The last inequality will be used at a later time. Assume now that est_i has been adjusted by applying $\gamma^{(2)}_{A,i}$. Note that this means that est_i is increased or remains unchanged. Thus, if the output condition is satisfied for the couple (A, i) *prior* the adjustment of est_i then it is satisfied *after* the adjustment, so that

$$\max_{u \in A \cup \{i\}, v \in A, u \neq v} \{lct_v - est^*_u\} < p(A \cup \{i\}) \tag{16.4.31}$$

still holds for $est^*_i := \max\{est_i, LB_2(A)\}$ and $est^*_u = est_u$ for all $u \neq i$. If we do not maximize over all but only a subset of values then we obtain a lower bound of the left side of this inequality and

$$\max_{u \in A, v \in V, u \neq v} \{lct_v - est^*_u\} < p(A \cup \{i\}) \,. \tag{16.4.32}$$

Rewriting $p(A \cup \{i\}) = p(V \cup \{i\}) + p(A - V)$ then leads to

$$\max_{u \in A, v \in V, u \neq v} \{lct_v - (est^*_u + p(A - V)\} < p(A \cup \{i\}) \,. \tag{16.4.33}$$

The left side of (16.4.33) can be simplified using the identity

$$\max_{u \in A, v \in V, u \neq v} \{lct_v - (est^*_u + p(A - V))\}$$
$$= \max_{v \in V} \{lct_v - (EST^*_{min}(A) + p(A - V))\}. \tag{16.4.34}$$

This is not apparent at once and requires some explanations. At first, the term on the left side of (16.4.34) seems to be less than or equal to the term on the right side, since $EST^*_{min}(A) \leq est^*_u$ for all $u \in A$. We now choose $u' \in A$ such that $est^*_{u'} = EST^*_{min}(A)$. If $u' \in V \subset A$ then $EST^*_{min}(V) = EST^*_{min}(A)$. Since the earliest start times of all tasks with indices in A did not change, this is a contradiction to (16.4.30). Thus, the left side of (16.4.34) assumes the maximal value for $u = u' \notin V$, and both terms are indeed identical. Therefore, (16.4.33) is equivalent to

$$\max_{v \in V} \{lct_v - (EST^*_{min}(A) + p(A - V))\} < p(V \cup \{i\}). \tag{16.4.35}$$

The left side of (16.4.35) can be approximated using (16.4.30) which tells us that for all $u \in V$:

$$est^*_u > EST^*_{min}(A) + p(A - V) \tag{16.4.36}$$

Likewise, we can deduce

$$est^*_i \geq LB_2(A) = EST^*_{min}(A) + p(A) > EST^*_{min}(A) + p(A - V) \,. \tag{16.4.37}$$

So, $EST^*_{min}(A) + p(A - V)$ in (16.4.35) can be replaced by est^*_u for all $u \in V \cup \{i\}$ which yields

$$\max_{u \in V \cup \{i\}, v \in V, u \neq v} \{lct_v - est^*_u\} < p(V \cup \{i\}) \tag{16.4.38}$$

Observe that this is nothing but the output condition for the couple (V, i).

Since $LB_2(V) = EST^*_{min}(V) + p(V) = LB_3(A)$, a subsequent application of $\gamma^{(2)}_{V,i}$ leads to the same domain reduction and the second case (2) applies. This completes our proof. □

Sequence Consistency Tests Revisited

It has already been mentioned that applying both sequence and domain consistency tests together can lead to better search space reductions. Quite evidently, any domain reductions deduced by Theorem 16.4.13 can lead to additional edge orientations deduced by Theorem 16.4.11. We will now discuss the case in which the inverse is also true.

Imagine a situation in which $A \rightarrow i$ can be deduced for a subset of tasks, but in which the output condition does not hold for the couple (A, i). Such a situation can actually occur as has, for instance, been shown in Example 16.4.8 for the three tasks T_i, T_j, T_k: while $j \rightarrow i$ and $k \rightarrow i$ can be separately deduced without, however, implying a domain reduction, the output condition fails for the couple $(\{j,k\}, i)$. This motivates the following obvious theorem as an extension of Theorem 16.4.13.

Theorem 16.4.17 (*Input/Output Domain Consistency Tests, part 2*).

Let $A \subsetneq B$ and $i \notin A$. If $A \rightarrow i$ then the earliest start time of task T_i can be adjusted to $est_i := \max\{est_i, LB_h(A)\}$, $h \in \{1,2,3,4\}$. □

Algorithms and Implementation Issues

An important question to answer now is whether there exist efficient algorithms that implement the input/output consistency tests. There are two obstacles which have to be overcome: the computation of the domain adjustments and the detection of the couples (A, i) which satisfy the input/output conditions.

Regarding the former, computing the non-preemptive bound is ruled out due to the NP-hardness result. At the other extreme, the "earliest completion time bound" (LB_1) is a too weak approximation. Therefore, only the "sum bound" (LB_2) or the preemptive bound (LB_3) remain candidates for the domain adjustments. Recall that both bounds are equivalent with respect to the induced Δ-fixed-point. Regarding the computational complexity, however, the two bounds are quite different: on the one hand, computing LB_2 requires linear time complexity $O(|A|)$ in contrast to the $O(|A|\log|A|)$ time complexity for computing LB_3. On the other hand, establishing the Δ-fixed-point, LB_2 usually has to be computed more often than LB_3, and it is not clear which factor - complexity of bound computation or number of iterations - dominates the other.

Let us turn to the second problem. An efficient implementation of the input/output consistency tests is obviously not possible if all pairs (A, i) of subsets

$A \subsetneq B$ and tasks T_i, $i \notin A$ are to be tested separately. Fortunately, it is not necessary to do so as has been first shown by Carlier and Pinson [CP90]. They developed an $O(n^2)$ algorithm (with $n = |B|$) which deduces all edge orientations and all domain reductions that are implied by the modified input/output conditions and the preemptive bound adjustment[3]. The fundamental idea was to test the modified input/output conditions and to compute the preemptive bound adjustments *simultaneously*. Several years later, Carlier and Pinson [CP94] and Brucker et al. [BJK94] presented $O(n \log n)$ algorithms which until now have the best asymptotic performance, but require quite complex data structures.

Nuijten [Nui94], Caseau and Laburthe [CL95] and Martin and Shmoys [MS96] have chosen a solely domain oriented approach and proposed different algorithms for implementing Theorem 16.4.13 based again on the modified input/output conditions. Nuijten developed an $O(n^2)$ algorithm which as well can be applied to scheduling problems with discrete resource capacity. Caseau and Laburthe presented an $O(n^3)$ algorithm based on the concept of *task sets* which works in an incremental fashion, so that $O(n^3)$ is a seldom worst case. The algorithm introduced by Martin and Shmoys [MS96] has a time complexity of $O(n^2)$.

An $O(n^3)$ algorithm which deduces all edge orientations implied by Theorem 16.4.11 has been derived by Phan Huy [Pha00]. He also presents an $O(n^2 \log n)$ for deriving all domain adjustments implied by Theorem 16.4.17.

16.4.5 Input/Output Negation Consistency Tests

In the last subsection, conditions have been described which imply that a task has to be processed before (after) another set of tasks. In this subsection, the inverse situation that a task *cannot* be processed first (last) is studied.

Sequence Consistency Tests

The following theorem is due to Carlier and Pinson [CP89]. For reasons near at hand, we have chosen the name input/output negation for the conditions described in this theorem.

Theorem 16.4.18 (*Input/Output Negation Sequence Consistency Tests*).

Let $A \subsetneq B$ and $i \notin A$. If the input negation condition

[3] It is common practice to only report the time complexity for applying all consistency tests *once*. In general, the number of iterations necessary for computing the Δ-fixed-point has to be considered as well. In the worst case, this accounts for an additional factor c which depends upon the size of the current domains. In practice, however, c is a rather small constant.

$$LCT_{max}(A) - est_i < p(A \cup \{i\}) \tag{16.4.39}$$

is satisfied then task T_i cannot be processed before all tasks $T_j, j \in A$. Likewise, if the output negation condition

$$lct_i - EST_{min}(A) < p(A \cup \{i\}) \tag{16.4.40}$$

is satisfied then task T_i cannot be processed after all other tasks $T_j, j \in A$.

Proof. If T_i is processed before $T_j, j \in A$ then all tasks with indices in A have to be processed within the time interval $[est_i, LCT_{max}(A))$. This leads to a contradiction if (16.4.39) is satisfied. The second assertion can be shown analogously. □

The input/output negation conditions are a relaxation of the input/output conditions and so are more often satisfied. However, the conclusions drawn in Theorem 16.4.18 are usually weaker than those drawn in Theorem 16.4.11, except for A contains a single task[4]. An important issue is therefore the development of strong domain reduction rules based on the limited information deduced.

Domain Consistency Tests

We will only study the input negation condition and the adjustments of earliest start times. Let us suppose that (16.4.39) is satisfied for $A \subsetneq B$ and $i \notin A$. Since, then, T_i cannot be processed before all tasks $T_j, j \in A$, there must be a task in A which starts and finishes before T_i, although we generally do not know which one. Thus, a lower bound of the earliest start time of T_i is

$$LB_5(A, i) = \min_{u \in A} \{ect_u\} \tag{16.4.41}$$

Caseau and Laburthe [CL95] made the following observation: if T_i cannot be processed first then, in any feasible schedule, there must exist a subset $\emptyset \neq V \subseteq A$, so that $V \rightarrow i \rightarrow A - V$. As a necessary condition, this subset V has to satisfy

$$LCT_{max}((A - V) \cup \{i\}) - EST_{min}(V) \geq p(A \cup \{i\}) . \tag{16.4.42}$$

Consequently, they proposed

$$LB_6(A, i) = \min_{\emptyset \neq V \subseteq A} \{ LB_2(V) \mid V \text{ satisifies (16.4.42)} \} \tag{16.4.43}$$

as a lower bound for the earliest start time of T_i. Notice, however, that if V satisfies (16.4.42) then the one-elemented set $V' := \{u\} \subseteq V$ with $est_u = EST_{min}(V)$ satisfies (16.4.42) as well. Further, $LB_2(V) = EST_{min}(V) + p(V) = est_u + p(V)$

[4] In this case, the input/output negation sequence consistency test coincides with the input/output sequence consistency test for pairs of operations.

$\geq est_u + p_u = LB_2(V')$, so that the expression in (16.4.43) is minimal for a one-element set. Therefore, setting $A_u := (A - \{u\}) \cup \{i\}$ we can rewrite

$$LB_6(A,i) = \min_{u \in A} \{ ect_u \mid LCT_{max}(A_u) - est_u \geq p(A_u \cup \{u\}) \} \tag{16.4.44}$$

This bound has a quite simple interpretation: the minimal earliest completion time is only chosen among all tasks which do not satisfy the input negation condition, because those who do, cannot start at the first position.

Up to now, est_i has been adjusted to the earliest completion time of some single task. The time bound adjustment can be improved if a condition is derived that detects a situation in which more than one task have to be processed before T_i. Observe that if for a subset $\emptyset \neq V \subseteq A$ the sequence $V \rightarrow i \rightarrow A - V$ is feasible then the following condition must hold:

$$LCT_{max}((A - V) \cup \{i\}) - est_i \geq p((A - V) \cup \{i\}) \,. \tag{16.4.45}$$

This implies the lower bounds on the earliest start time:

$$LB_7(A,i) := \min_{\emptyset \neq V \subseteq A} \{ LB_2(V) \mid V \text{ satisfies (16.4.45)} \} \tag{16.4.46}$$

$$LB_8(A,i) := \min_{\emptyset \neq V \subseteq A} \{ LB_3(V) \mid V \text{ satisfies (16.4.45)} \} \tag{16.4.47}$$

Finally, we can try to find the exact earliest start time of task T_i by computing

$$LB_9(A,i) := \min_{\emptyset \neq V \subseteq A} \{ LB_4(V) \mid V \rightarrow i \rightarrow A - V \text{ is feasible} \} \,. \tag{16.4.48}$$

The following theorem which is a generalization of the consistency test (16.4.20) summarizes the results derived above.

Theorem 16.4.19 (*Input/Output Negation Domain Consistency Tests*).

If the input negation condition is satisfied for $A \subsetneq B$ and $i \notin A$ then the earliest start time of task T_i can be adjusted to $est_i := \max\{est_i, LB_h(A,i)\}$, $h \in \{5,6,7,8,9\}$.

Dominance Relations

For $h \in \{5,6,7,8,9\}$, let $\Gamma_{\neg in}(h) := \{ \gamma_{A,i}^{(h)} \mid A \subsetneq B,\, i \notin A \}$ denote the set of input negation domain consistency tests defined in Theorem 16.4.19:

$$\gamma_{A,i}^{(h)} : LCT_{max}(A) - est_i < p(A \cup \{i\}) \;\Rightarrow\; est_i := \max\{est_i, LB_h(A,i)\} \,.$$

Lemma 16.4.20

The following dominance relations hold:

1. $\Gamma_{\neg in}(5) \trianglelefteq \Gamma_{\neg in}(6) \trianglelefteq \Gamma_{\neg in}(9)$,

2. $\Gamma_{\neg in}(5) \leqq \Gamma_{\neg in}(7) \leqq \Gamma_{\neg in}(8) \leqq \Gamma_{\neg in}(9)$.

Lemma 16.4.21

$$\Gamma_{\neg in}(5) \sim \Gamma_{\neg in}(6).$$

Proof. We only have to prove that $\Gamma_{\neg in}(6) \leqq \Gamma_{\neg in}(5)$. It is sufficient to show that for all $A \subsetneq B$, $i \notin A$ and $\Delta \in \Theta$, there exist $A^1, \cdots, A^r \subsetneq B$ such that

$$(\gamma^{(5)}_{A^r,i} \circ \cdots \circ \gamma^{(5)}_{A^1,i})(\Delta) \subseteq \gamma^{(6)}_{A,i}(\Delta) \tag{16.4.49}$$

For the sake of simplicity, we omit an exact proof but only describe the basic ideas. Let $U \subseteq A$ denote the index set of tasks satisfying the input negation condition, i.e. $U := \{ u \in A \mid LCT_{max}(A_u) - est_u < p(A_u \cup \{u\})\}$ with $A_u := (A - \{u\}) \cup \{i\}$.

Recall that

(i) $LB_5(A,i) = \min_{u \in A}\{ect_u\}$,

(ii) $LB_6(A,i) = \min_{u \in A-U}\{ect_u\}$.

If both bounds are identical then, obviously, $\gamma^{(6)}_{A,i}(\Delta) = \gamma^{(5)}_{A,i}(\Delta)$. This identity, for instance, holds if U is empty. Thus, in the following, we restrict our attention to the case $|U| > 0$. If $u \in A$ is a task satisfying $ect_u = LB_5(A,i) < LB_6(A,i)$ then $u \in U$ and

$$est_u + p(A_u \cup \{u\}) = ect_u + p(A_u) > LCT_{max}(A_u) .$$

If the earliest start time of T_i has been adjusted to $est^*_i := \max\{est_i, LB_5(A,i)\}$ by applying $\gamma^{(5)}_{A,i}$ then we have $est^*_i \geq ect_u$, so

$$est^*_i + p(A_u) > LCT_{max}(A_u) \geq LCT_{max}(A_u - \{i\})$$

or

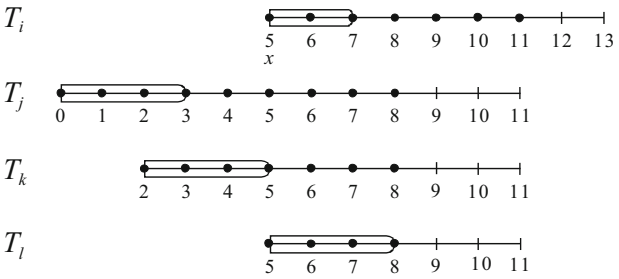
$$est^*_i + p((A - \{u\}) \cup \{i\}) > LCT_{max}(A - \{u\})$$

which is the input negation condition for the couple $(A - \{u\}, i)$. Therefore, est^*_i can be adjusted once more to $LB_5(A - \{u\}, i)$. If $LB_5(A - \{u\}, i) = LB_6(A - \{u\}, i)$ then we are done, since $LB_6(A - \{u\}, i) \geq LB_6(A, i)$. Otherwise, we are in the same situation as above which allows us to continue in the same manner. Finally, observe that the number of adjustments is finite and bounded by $|A|$. □

Example 16.4.22

Consider the example shown in Figure 16.4.6 with four tasks indexed as i, j, k, l. A closer look at the set of feasible schedules reveals that δ_j, δ_k and δ_l cannot be reduced. Likewise, it can be seen that i cannot be the input of $A = \{j, k, l\}$ which

is detected by the input negation condition, since $LCT_{max}(A) - est_i = 11 - 5 < 11 = p(A \cup \{i\})$. Using LB_5, no time bound adjustment is possible, since $LB_5(A, i) = 3$. However, there exists no feasible schedule in which only one task is processed before T_i. Indeed, $LB_7(A, i) = 6$ leads to a stronger time bound adjustment. After the domain reduction, a fixed point is reached, so this example and Lemma 16.4.20 prove that $\Gamma_{\neg in}(5) \prec \Gamma_{\neg in}(7)$. □



sequence	st_i	st_j	st_k	st_l
$j \to k \to i \to l$	6	0	3	8
$j \to k \to l \to i$	9	0	3	6
$j \to l \to k \to i$	11	0	8	5
$k \to j \to l \to i$	11	5	2	8
$k \to l \to j \to i$	11	8	2	5

Figure 16.4.6 *Comparing* $\Gamma_{\neg in}(5)$ *and* $\Gamma_{\neg in}(7)$.

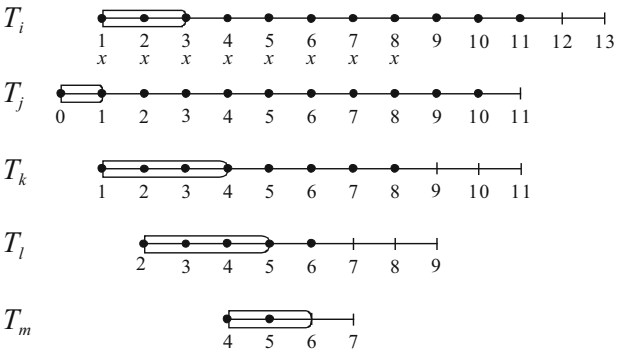
Lemma 16.4.23

$$\Gamma_{\neg in}(7) \sim \Gamma_{\neg in}(8) .$$

Proof. Similar to Theorem 16.4.16. □

Example 16.4.24

Consider the situation in Figure 16.4.7 with five tasks indexed as i, j, k, l, m. Again, δ_j, δ_k, δ_l and δ_m cannot be reduced. Further, it can be seen that i is the output of $A = \{j, k, l, m\}$ with the earliest start time being $LB_9(A, i) = 9$. However, the output condition is not satisfied for the couple (A, i). The input negation condition holds, since $LCT_{max}(A) - est_i = 11 - 1 < 11 = p(A \cup \{i\})$, but $LB_h(A, i) = 1$ for all $h \in \{5, 6, 7, 8\}$. Thus, the current domain of T_i remains unchanged if these bound adjustments are applied, i.e. a fixed point is reached. This and Lemma 16.4.20 prove the relation $\Gamma_{\neg in}(8) \prec \Gamma_{\neg in}(9)$. □



sequence	st_i	st_j	st_k	st_l	st_m
$j \to k \to m \to l \to i$	9	0	1	6	4
$j \to l \to m \to k \to i$	10	0	7	2	5
$k \to m \to l \to j \to i$	10	9	1	6	4
$l \to m \to j \to k \to i$	11	7	8	2	5
$l \to m \to k \to j \to i$	11	10	7	2	5

Figure 16.4.7 Comparing $\Gamma_{\neg in}(8)$ and $\Gamma_{\neg in}(9)$.

Altogether, we have proven the following theorem.

Theorem 16.4.25 (*dominance relations for input negation consistency tests*).

$$\Gamma_{\neg in}(5) \sim \Gamma_{\neg in}(6) \prec \Gamma_{\neg in}(7) \sim \Gamma_{\neg in}(8) \prec \Gamma_{\neg in}(9) \,. \qquad \square$$

Algorithms and Implementation Issues

Input negation consistency tests which use the “simple earliest completion time bound" (LB_5) as time bound adjustment and their output negation counterparts have been applied by Nuijten [Nui94], Baptiste and Le Pape [BL95] and Caseau and Laburthe [CL95]. Caseau and Laburthe have integrated the tests in their scheduling environment based on task sets in a straightforward manner which yields an algorithm with time complexity $O(n^3)$. All these algorithms only test some, but not all interesting couples (A, i). An algorithm which deduces all domain reductions with time complexity $O(n^2)$ has only been developed by Baptiste and Le Pape [BL96]. A similar implementation is proposed by Phan Huy in [Pha00]. Nuijten and Le Pape [NL98] derived several consistency tests which are similar to the input/output negation consistency tests with the time bound adjustment LB_8 and can be implemented with time complexity $O(n^2 \log n)$ and $O(n^3)$ respectively.

16.4.6 Input-or-Output Consistency Tests

In this subsection, some new consistency tests are presented which are not subsumed by the consistency tests presented in the previous subsections. They are based on the input-or-output conditions which have been introduced by Dorndorf et al. [DPP99].

Domain and Sequence Consistency Tests

The input-or-output conditions detect situations in which either (a) a task T_i has to be processed first or (b) a task T_j has to be processed last within a set of tasks. There exists a sequence and a domain oriented consistency test based on the input-or-output condition. Both tests are summarized in the next theorem.

Theorem 16.4.26 (*input-or-output consistency tests*).

Let $A \subsetneq B$ and $i, j \notin A$. If the input-or-output condition

$$\max_{u \in A \cup \{j\}, v \in A \cup \{i\}, u \neq v} \{lct_v - est_u\} < p(A \cup \{i,j\}) \tag{16.4.50}$$

is satisfied then either task T_i has to be processed first or task T_j has to be processed last within $A \cup \{i,j\}$. If $i \neq j$ then task T_i has to be processed before T_j and the domains of T_i and T_j can be adjusted as follows:

$$est_j := \max\{est_j\,,\, est_i + p_i\}\,,$$

$$lst_j := \min\{lst_i\,,\, lst_j - p_i\}\,.$$

Proof. If T_i is neither processed before, nor T_j processed after all other tasks in $A \cup \{i,j\}$ then all tasks in $A \cup \{i,j\}$ have to be processed within a time interval of maximal size

$$\max_{u \in A \cup \{j\}, v \in A \cup \{i\}, u \neq v} \{lct_v - est_u\}.$$

This is a contradiction to (16.4.50).

Now, since T_i has to be processed first or T_j processed last within $A \cup \{i,j\}$, we can deduce that T_i has to be processed before T_j if $i \neq j$. This immediately implies the domain deductions described above. □

By substituting (16.4.50) with

$$LCT_{max}((A \cup \{i\}) - EST_{min}(A \cup \{j\}) < p(A \cup \{i,j\})\,, \tag{16.4.51}$$

we obtain the *modified input-or-output conditions* which can be tested more easily, but are less often satisfied than the input-or-output conditions.

Example 16.4.27

In Figure 16.4.8 an example for the application of the input-or-output consistency tests with four tasks indexed as i, j, k, l is shown.
Since

$$\max_{u\in\{j,k,l\},v\in\{i,k,l\},u\neq v}\{lct_v - est_u\} = 6 < 7 = p(\{i,j,k,l\})$$

we can conclude that T_i has to be processed before T_j. Thus, we can adjust $est_j := 4$ and $lst_i := 4$. □

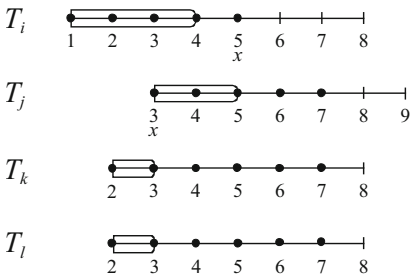


Figure 16.4.8 *The input-or-output consistency test.*

Algorithms and Implementation Issues

Deweß [Dew92] and Brucker et al. [BJK94] discuss conditions which examine all permutations of a fixed length r and which are thus called r-set conditions. Brucker et al. [BJK94] developed an $O(n^2)$ algorithm for testing all 3-set conditions which is equivalent to testing all input-or-output conditions for triples of tasks. Phan Huy [Pha00] developed an $O(n^3)$ algorithm for deriving all edge orientations implied by the modified input-or-output conditions. This algorithm can be generalized to an $O(n^4)$ algorithm which deduces all edge orientations implied by the input-or-output conditions.

16.4.7 Energetic Reasoning

The conditions described in the previous subsections for testing consistency were all founded on the principle of comparing a time interval in which a set of tasks A has to be processed with the total processing time $p(A)$ of these tasks. The time intervals chosen were defined through the earliest start and latest completion times of some of the tasks. This fundamental principle can be generalized by considering arbitrary time intervals $[t_1, t_2)$, on the one hand, and replacing simple processing time $p(A)$ with interval processing time $p(A, t_1, t_2)$, on the other hand. Erschler et al. [ELT91], see also [LEE92], were the first to introduce this idea under the name of *energetic reasoning*. Indeed, the interval processing time can

be interpreted as resource energy demand which encounters a limited resource energy supply that is defined through the time interval. The original concept of Erschler et al. considered cumulative scheduling problems with discrete resource capacity. Their results have been improved by Baptiste and Le Pape [BL95] for disjunctive constraints. We will take a closer look at these results and compare them to the consistency tests described so far.

Interval Processing Time

Let us first define the interval processing time of a task T_i for a given time interval $[t_1, t_2)$, $t_1 < t_2$. The interval processing time $p_i(t_1, t_2)$ is the smallest amount of time during which T_i has to be processed within $[t_1, t_2)$. Figure 16.4.9 shows four possible situations: (1) T_i can be completely contained within the interval, (2) overlap the entire interval, (3) have a minimum processing time in the interval when started as early as possible or (4) have a minimum processing time when started as late as possible. The fifth situation not depicted applies whenever, given the current domains, T_i does not necessarily have to be processed within the given time interval. Consequently,

$$p_i(t_1, t_2) := \max\{\, 0, \min\{p_i, t_2 - t_1, ect_i - t_1, t_2 - lst_i\}\}. \tag{16.4.52}$$

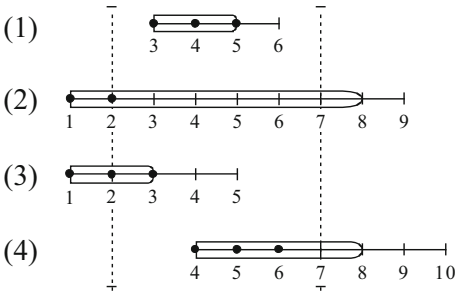


Figure 16.4.9 *Types of relations between a task and a time interval.*

The interval processing time of a subset of tasks A is given by $p(A, t_1, t_2) := \sum_{i \in A} p_i(t_1, t_2)$. Finally, let $B_{(t_1,t_2)} := \{\, i \in B \mid p_i(t_1, t_2) > 0 \,\}$ denote the set of tasks which have to be processed completely or partially within $[t_1, t_2)$.

Energetic Input/Output Consistency Tests

Baptiste and Le Pape [BL95] examined situations in which the earliest start time of a task T_i can be updated using the concept of interval processing times. Assume, for instance, that T_i finishes before t_2. The interval processing time of T_i in

$[t_1, t_2)$ would then be $p_i'(t_1, t_2) = \min\{p_i, t_2 - t_1, ect_i - t_1\}$.[5] However, if $t_2 - t_1 < p(B - \{i\}, t_1, t_2) + p_i'(t_1, t_2)$ then the assumption cannot be true, so that T_i has to finish after t_2. Baptiste and Le Pape showed that est_i can be then updated to

$$est_i := \max\{est_i, t_1 + p(B - \{i\}, t_1, t_2)\}. \tag{16.4.53}$$

A stronger domain reduction rule is presented in the following theorem.

Theorem 16.4.28 *Energetic output conditions.*

Let $i \in B$ and $t_1 < t_2$. If the energetic output condition

$$t_2 - t_1 < p(B - \{i\}, t_1, t_2) + \min\{p_i, t_2 - t_1, ect_i - t_1\} \tag{16.4.54}$$

is satisfied then $B_{(t_1,t_2)} - \{i\}$ is not empty, and T_i has to be processed after all tasks of $B_{(t_1,t_2)} - \{i\}$. Consequently, est_i can be adjusted to $est_i := \max\{est_i, LB_h(B_{(t_1,t_2)} - \{i\})\}$, $h \in \{1, 2, 3, 4\}$.

Proof. If (16.4.54) is satisfied then $p(B - \{i\}, t_1, t_2) > 0$ and $B_{(t_1,t_2)} - \{i\}$ is not empty. Furthermore, T_i must finish after t_2. By definition, all tasks in $B_{(t_1,t_2)} - \{i\}$ have positive processing times in the interval $[t_1, t_2)$ and so must start and finish before T_i. This proves $B_{(t_1,t_2)} - \{i\} \rightarrow i$ from which follows the domain reduction rule. □

Energetic input conditions can be defined in a similar way. Observe that the domain adjustment in Theorem 16.4.28 is stronger than the one defined in (16.4.53) if the "sum bound" (LB_2) or a stronger bound is used. We omit the simple proof due to the observations made in the following.

Up to now, it remained an open question which time intervals were especially suited for testing the energetic input/output conditions in order to derive strong domain reductions. We will sharpen this question and ask whether Theorem 16.4.28 really leads to stronger domain reductions at all if compared with other known consistency tests. Quite surprisingly, the answer is “no".

Theorem 16.4.29 (*comparing output and energetic output conditions*).

If the energetic output condition

$$t_2 - t_1 < p(B - \{i\}, t_1, t_2) + \min\{p_i, t_2 - t_1, ect_i - t_1\}$$

is satisfied for a task T_i, $i \in B$ and the time interval $[t_1, t_2)$ then the output condition

$$\max_{u \in A \cup \{i\}, v \in A, u \neq v} \{lct_v - est_u\} < p(A \cup \{i\})$$

[5] Here and later, we will assume that $p_i'(t_1, t_2) \geq 0$ which is not a serious restriction.

is satisfied for the couple $(B_{(t_1,t_2)} - \{i\}, i)$.

Proof. If the energetic output condition is satisfied then $B_{(t_1,t_2)} - \{i\}$ is not empty, and there exists a task T_v with $v \in B_{(t_1,t_2)} - \{i\}$. Let us first consider the case $u \in B_{(t_1,t_2)} - \{i\}$, $u \neq v$. We can approximate the right side of (16.4.54) and obtain

$$\begin{aligned} t_2 - t_1 &< p(B - \{i\}, t_1, t_2) + p_i \\ &= p(B - \{i,u,v\}, t_1, t_2) + p_u(t_1, t_2) + p_v(t_1, t_2) + p_i . \end{aligned} \tag{16.4.55}$$

Since u, $v \in B_{(t_1,t_2)}$, we know from (16.4.52) that $t_2 - lst_v \geq p_v(t_1, t_2)$ and $ect_u - t_1 \geq p_u(t_1, t_2)$, and we can approximate

$$t_2 - t_1 < p(B - \{i,u,v\}, t_1, t_2) + ect_u - t_1 + t_2 - lst_v + p_i \tag{16.4.56}$$

which is equivalent to

$$lst_v - ect_u < p(B - \{i,u,v\}, t_1, t_2) + p_i . \tag{16.4.57}$$

Note that $p(B - \{i,u,v\}, t_1, t_2) \leq p(B_{(t_1,t_2)} - \{i,u,v\})$, so we arrive at

$$lst_v - ect_u < p(B_{(t_1,t_2)} - \{u,v\}) . \tag{16.4.58}$$

or, equivalently,

$$lct_v - est_u < p(B_{(t_1,t_2)}) . \tag{16.4.59}$$

Now, consider the case $u = i \neq v$. Using (16.4.54), we have

$$\begin{aligned} t_2 - t_1 &< p(B - \{i\}, t_1, t_2) + ect_i - t_1 \\ &= p(B - \{i,v\}, t_1, t_2) + p_v(t_1, t_2) + ect_i - t_1 . \end{aligned} \tag{16.4.60}$$

We can, again, substitute $p_v(t_1, t_2)$ with $t_2 - lst_v$ and obtain

$$lst_v - ect_i < p(B - \{i,v\}, t_1, t_2) . \tag{16.4.61}$$

A similar line of argumentation as above leads to

$$lct_v - est_i < p(B_{(t_1,t_2)}) . \tag{16.4.62}$$

Finally, combining (16.4.59) and (16.4.62) leads to the output condition for the couple $(B_{(t_1,t_2)} - \{i\}, i)$ which proves our assertion. □

A similar result applies for the energetic input condition. Inversely, a quite simple proof which is omitted shows that the input/output conditions are subsumed by the energetic generalizations, so that both concepts are in fact equivalent.

Other Energetic Consistency Tests

It is possible to derive input/output negation conditions and input-or-output conditions that are based on energetic reasoning. However, as in the case of the input/output conditions, they do not imply additional domain reductions which are not also deduced by the corresponding non-energetic conditions. We therefore omit a detailed presentation of these conditions.

The results of this subsection have an important implication. They tell us that for the disjunctive scheduling problem, all known consistency tests that are based on energetic reasoning are not more powerful than their non-energetic counterparts. It is not clear whether this holds for arbitrary consistency tests, although we strongly assume this. A step towards proving this conjecture has been made in [DPP99] where it has been shown that, regardless of the chosen consistency tests, the interval processing times $p(A, t_1, t_2)$ can always be replaced by the simple processing times $p(A)$.

16.4.8 Shaving

All consistency tests presented so far share the common idea that a possible start time st_i of a task T_i can be removed from its current domain δ_i if there exists no feasible schedule in which T_i actually starts at that time. In this context, the consistency tests that have been introduced in the Sections 16.4.3 through 16.4.7 can be interpreted as sufficient conditions for proving that no feasible schedule can exist which involve a specific start time assignment st_i. In Section 16.4.3, for instance, we have tested the sufficient condition whether there exists a 2- or 3-feasible start time assignment.

This general approach has been summarized by Martin and Shmoys under the name *shaving* [MS96]. They proposed additional shaving variants. *Exact one-machine shave* verifies whether a non-preemptive schedule exists by solving an instance of the one-machine scheduling problem in which the start time $st_i \in \{est_i, lst_i\}$ is fixed. Quite obviously, exact one-machine shave is NP-hard and equivalent to establishing *n-b*-consistency. *One-machine shave* relaxes the non-preemption requirement and searches for a (possibly) preemptive schedule.

Carlier and Pinson [CP94] and Martin and Shmoys [MS96] independently proposed the computation of Δ-fixed-points as a method for proving the non-existence of a feasible schedule. Given a set of consistency tests Γ and a set of current domains, say Δ', a feasible schedule cannot exist if a current domain in $\Gamma(\Delta')$ is empty. Carlier and Pinson, and Martin and Shmoys who coined the name *C-P shave* have chosen the modified input/output domain consistency tests and the precedence consistency tests as underlying set of consistency tests. Martin and Shmoys have further proposed *double shave* which applies C-P shave for detecting inconsistencies. Torres and Lopez [TL00] review possible extensions of shaving techniques that have been proposed for job shop scheduling. Dorndorf

et al. [DPP01] very successfully apply shaving techniques to the open shop scheduling problem (OSP), which is a special case of the DSP (cf. Chapter 9).

16.4.9 A Comparison of Disjunctive Consistency Tests

Let us summarize the results derived so far. In Figure 16.4.10, the dominance relations between different levels of bound-consistency and classes of consistency tests are shown[6]. A strict dominance is represented by an arc $\rightarrow$, while $\leftrightarrow$ stands for an equivalence relation. An encircled "+" means that the corresponding classes of consistency tests taken together imply a dominance relation. Since the dominance relation is transitive, we do not display all relations explicitly.

Let us start with the upper half of the figure. Obviously, n-b-consistency and exact one-machine shave are equivalent and strictly dominate all other consistency tests. On the left side, n-b-consistency, of course, subsumes all levels of k-b-consistency for $k \leq n$.

In the center of the figure, the consistency tests with an input/output component in their names are shown. As has been proven in Section 16.4.7, the energetic consistency tests are equivalent to the non-energetic ones. In Example 16.4.12, we have verified that the input/output consistency tests dominate the modified input/output consistency tests. The same dominance relation holds for the input-or-output tests when compared to the modified tests. In Section 16.4.3 we have shown that the input/output and input/output negation consistency tests taken together establish strong 3-b-consistency if for the former the "sum bound" (LB_2) and for the latter the "simple earliest completion time bound" (LB_5) are applied for adjusting the current domains. The input/output and input/output negation tests usually imply more than 3-b-consistency as can be seen in Example 16.4.15. However, if only pairs and triples of tasks are considered then the equivalence relation holds. Further, it has been shown in Section 16.4.3 that applying the input/output consistency tests for pairs of tasks is equivalent to establishing 2-b-consistency if the "earliest completion time bound" (LB_1) is used as time bound adjustment.

Let us now turn to the right side of the figure. It is not hard to show that double shave strictly dominates C-P shave which in turn strictly dominates one-machine shave. Apart from this, there exists no particular relationship between double shave and C-P shave and the other consistency tests. However, double shave and C-P shave usually lead to significantly stronger domain reductions as has been verified empirically. Finally, Martin and Shmoys [MS96] have shown that one-machine shave is equivalent to the modified input/output domain consistency tests.

[6] Although the dominance relation has only been defined for sets of consistency tests, it can be extended in a straightforward manner to the levels of bound-consistency.

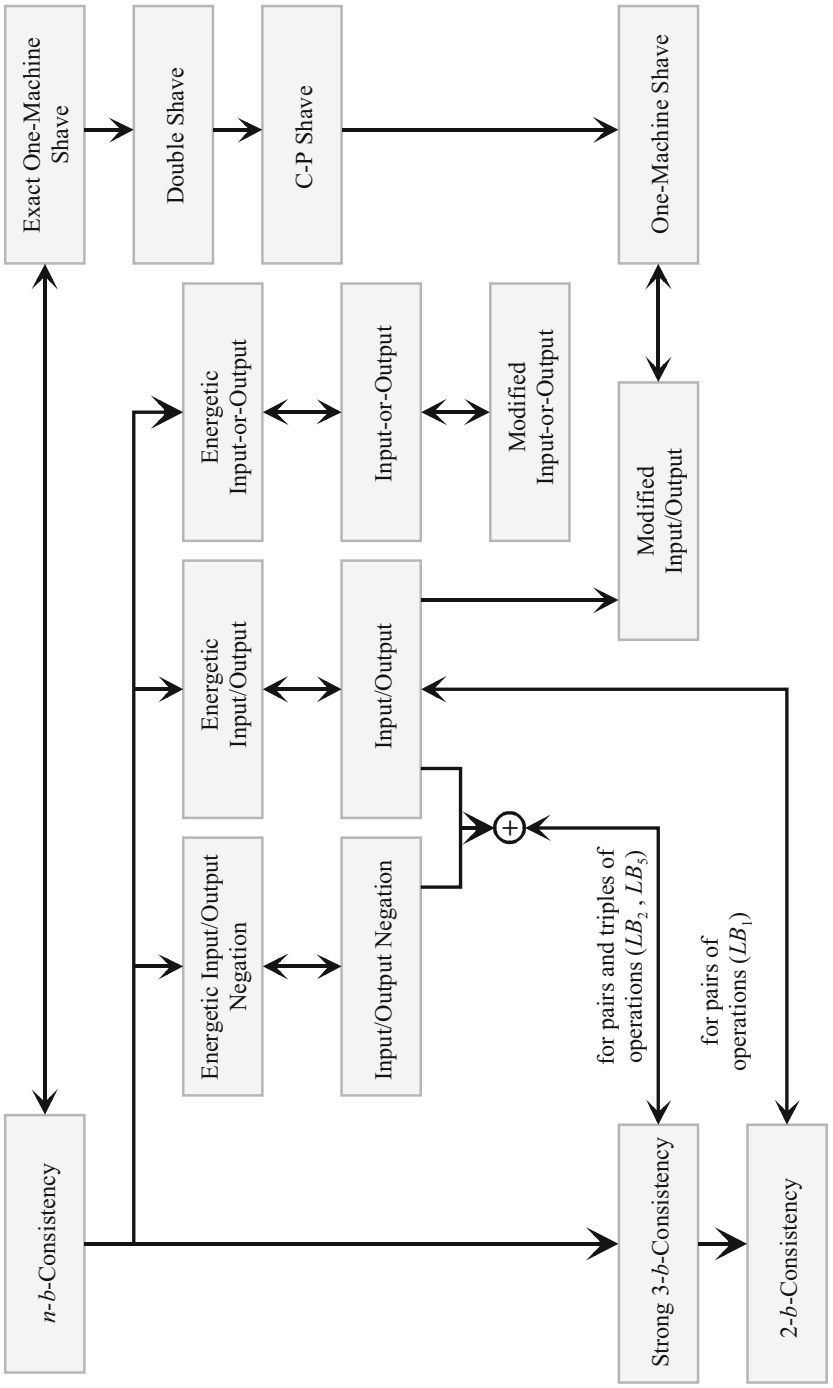


Figure 16.4.10 *Dominance relations*.

16.4.10 Precedence vs. Disjunctive Consistency Tests

The consistency tests which have been developed for the disjunctive constraints can be applied to an instance of the DSP by decomposing this instance into (preferably maximal) cliques. Since all consistency tests presented are monotonous, they can be applied in an arbitrary order and always result in the same Δ-fixed-point. However, the runtime behaviour differs extremely depending on the order of application that has been chosen.

An ordering rule which has been proven to be quite effective is to perform the sequence consistency tests that are likely to deduce more edge orientations and have a lower time complexity in the beginning. A particular set of consistency tests is only triggered if all "preceding" consistency tests do not imply any deductions any more. This ensures that the more costly consistency tests are only seldomly applied and contribute less in the overall computational costs.

Finally, Nuijten and Sourd [NS00] have recently described consistency checking techniques for the DSP that are based on the simultaneous consideration of precedence constraints and disjunctive constraints.

16.5 Conclusions

Constraint propagation is an elementary method which reduces the search space of a search or optimization problem by analyzing the interdependencies between the variables, domains and constraints that define the set of feasible solutions. Instead of achieving full consistency with respect to some concept of consistency, we generally have to content ourselves with approximations due to reasons of complexity. In this context, we have evaluated classical and new consistency tests for the DSP which are simple rules that reduce the domains of variables (domain consistency tests) or derive knowledge in a different form, e.g. by determining the processing sequences of a set of tasks (sequence consistency tests).

The particular strength of this approach is based on the repeated application of the consistency tests, so that the knowledge derived is propagated, i.e. reused for acquiring additional knowledge. The deduction of this knowledge can be described as the computation of a fixed point. Since this fixed point depends upon the order of the application of the consistency tests, Dorndorf et al. [DPP00] at first have derived a necessary condition for its uniqueness and have developed a concept of dominance which enables to compare different consistency tests. With respect to this dominance relation, they have examined the relationship between several concepts of consistency (bound-consistency, energetic reasoning and shaving) and the most powerful consistency tests known as the input/output, input/output negation and input-or-output consistency tests. They have been able to improve the well-known result that the input/output consistency tests for pairs of tasks imply 2-*b*-consistency by deriving the tests which establish strong 3-*b*-consistency. These consistency tests are slightly stronger than the famous ones

derived by Carlier and Pinson [CP89, CP90]. Dorndorf et al. [DPP00] have analyzed the input/output, input/output negation and input-or-output consistency tests and have classified different lower bounds which are used for the reduction of domains. They have shown that apparently weaker bounds still induce the same fixed point. Finally, an open question regarding the concept of energetic reasoning has been answered. In contrast to scheduling problems with discrete resource supply, they have shown that the known consistency tests based on energetic reasoning are equivalent to the tests based on simple processing times.

16.6 Appendix: Bound Consistency Revisited

In this section, we derive the time bound adjustments for establishing 3-*b*-consistency as has been announced in Section 16.4.3. Let us assume that the following condition

$$\begin{aligned}
&(\max\{lct_j - est_i, lct_k - est_i\} \geq p_i + p_j + p_k) \quad \vee && (i+ii)\\
&(est_j + p_j \leq est_i \quad \wedge \quad est_i + p_i \leq lst_k) \quad \vee && (iii)\\
&(est_k + p_k \leq est_i \quad \wedge \quad est_i + p_i \leq lst_j) \quad \vee && (iv)\\
&(est_i \geq \max\{\min\{est_j, est_k + p_j + p_k, est_j + p_j, est_k + p_k\}) && (v+vi)
\end{aligned} \tag{16.6.1}$$

is not satisfied given the current earliest and latest start times. As already mentioned, there exist two cases. In the first case, increasing est_i will never satisfy conditions ($i + ii$), (iii) and (iv). Therefore, we have to adjust est_i so as to satisfy condition (v+ vi). In the second case, condition ($i + ii$) is not satisfiable, but increasing est_i eventually satisfies (iii), (iv) or (v+ vi). Here, the minimal earliest start time for which (iii) or (iv) holds is not greater than the minimal earliest start time for which (v+ vi) holds. This will be proven in the remainder of this subsection.

We will first deal with the problem of how to distinguish between the two cases. The corresponding time bound adjustments will then be derived at a later time. In Lemma 16.6.1, a necessary and sufficient condition for the existence of $est_i^* \geq est_i$ satisfying condition (iii) is described.

Lemma 16.6.1 (*condition* (iii)).

There exists $est_i^* \geq est_i$ *such that condition* (iii) *is satisfied iff*

$$\max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k . \tag{16.6.2}$$

The smallest start time which then satisfies (iii) *is* $est_i^* = \max\{est_i, est_j + p_j\}$.

Proof. If condition (*iii*) is satisfied for $est_i^* \geq est_i$ then $est_j + p_j \leq est_i^*$ and $est_i^* + p_i \leq lst_k$, so that $\max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k$. This proves the direction $\Rightarrow$. In order to show $\Leftarrow$, let $\max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k$. If $est_i < est_j + p_j$ then $est_i^* = est_j + p_j$ is the smallest value which satisfies (*iii*). Otherwise, if $est_i \geq est_j + p_j$ then $est_i^* = est_i$ is the smallest value which satisfies (*iii*). □

Changing the roles of j and k in Lemma 16.6.1 leads to a similar result for condition (*iv*).

Corollary 16.6.2 (*conditions* (*iii*) *and* (*iv*))*.*

There exists $est_i^* \geq est_i$ *which satisfies* (*iii*) *or* (*iv*) *iff*

$$\begin{array}{l}(\max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k) \quad \vee \\ (\max\{est_k + p_k + p_i, est_i + p_i\} \leq lst_j)\end{array} \tag{16.6.3}$$

If Δ *is* 2-*b*-*consistent then* (16.6.3) *is equivalent to*

$$\begin{array}{l}(est_j + p_j + p_i \leq lst_k \ \vee \ est_k + p_k + p_i \leq lst_j) \quad \wedge \\ (est_i + p_i \leq lst_k \ \vee \ est_i + p_i \leq lst_j)\end{array} \tag{16.6.4}$$

Proof. The first assertion follows directly from Lemma 16.6.1. Let us show the second equivalence and assume that 2-*b*-consistency is established. Obviously, (16.6.3) immediately implies (16.6.4). The other direction, however, is not apparent at once.

Hence, let (16.6.4) be satisfied. It is sufficient to study the case $est_j + p_j + p_i \leq lst_k$, since $est_k + p_k + p_i \leq lst_j$ leads to a similar conclusion. Given (16.6.4), we can deduce that $est_i + p_i \leq lst_k$ or $est_i + p_i \leq lst_j$ $(*)$.

Now, if $est_i + p_i \leq lst_k$ then the first condition $\max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k$ of (16.6.3) is satisfied. If, however, $est_i + p_i > lst_k$ then 2-*b*-consistency implies $est_k + p_k \leq est_i$. Further, $est_i + p_i \leq lst_j$ due to $(*)$. Therefore, $est_k + p_k + p_i \leq lst_j$, and the second condition $\max\{est_k + p_k + p_i, est_i + p_i\} \leq lst_j$ of (16.6.3) is satisfied. □

Given these results, it is now quite easy to describe the adjustments of the earliest start times.

Lemma 16.6.3 (*adjusting earliest start times*, *part 1*)*.*

Let Δ *be* 2-*b*-*consistent. If*

$$\max_{u \in \{j,k\}, v \in \{i,j,k\}, u \neq v} \{lct_v - est_u\} < p_i + p_j + p_k \tag{16.6.5}$$

or

$$est_i + p_i > \max\{lst_j, lst_k\} \tag{16.6.6}$$

then (*i*+*ii*), (*iii*), (*iv*) *are not satisfiable for any* $est_i^* \geq est_i$. *The minimal earliest start time* $est_i^* \geq est_i$ *satisfying* (*v*+*vi*) *is then defined by*

$$est_i^* := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k, est_j + p_j, est_k + p_k\} . \quad (16.6.7)$$

Proof. We have shown in Lemma 16.4.5 that there exists no $est_i^* \geq est_i$ satisfying condition $(i + ii)$ iff

$$\max_{v \in \{j,k\}} \{lct_v - est_i\} < p_i + p_j + p_k . \quad (16.6.8)$$

Likewise, we have shown in Lemma 16.6.1 that there exists no $est_i^* \geq est_i$ satisfying condition (*iii*) or (*iv*) iff (16.6.4) is not satisfied, i.e. iff

$$\begin{array}{l}(est_j + p_j + p_i > lst_k \ \wedge \ est_k + p_k + p_i > lst_j) \ \vee \\ (est_i + p_i > lst_k \ \wedge \ est_i + p_i > lst_j)\end{array} \quad (16.6.9)$$

which is equivalent to

$$\begin{array}{l}(lct_k - est_j < p_i + p_j + p_k \ \wedge \ lct_j - est_k < p_i + p_j + p_k) \ \vee \\ est_i + p_i > \max\{lst_j, lst_k\}) .\end{array} \quad (16.6.10)$$

(16.6.8) and (16.6.10) together imply that (*i* + *ii*), (*iii*) and (*iv*) are not satisfiable, so we have to choose the minimal earliest start time est_i^* satisfying condition (*v* + *vi*) which leads to

$$est_i^* := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k, est_j + p_j, est_k + p_k\} . \quad (16.6.11)$$

It remains to combine (16.6.8) and (16.6.10) to one single condition. Making use of the fact that $est_i + p_i > \max\{lst_j, lst_k\}$ already implies (16.6.8), we can deduce that these two conditions are equivalent to:

$$\Big(\max_{u \in \{j,k\}, v \in \{i,j,k\}, u \neq v} \{lct_v - est_u\} < p_i + p_j + p_k\Big) \ \vee \ \big(est_i + p_i > \max\{lst_j, lst_k\}\big) .$$

This completes the proof. □

Lemma 16.6.4 (*adjusting earliest start times*, *part 2*).

Let Δ *be* 2*-b-consistent*. *If* (16.6.5) *and* (16.6.6) *are not satisfied but*

$$\max_{u \in \{j,k\}} \{lct_i - est_u\} < p_i + p_j + p_k \quad (16.6.12)$$

then (*i* + *ii*) *is not satisfiable for any* $est_i^* \geq est_i$. *The minimal earliest start time* $est_i^* \geq est_i$ *satisfying* (*iii*), (*iv*) *or* (*v* + *vi*) *is then defined through*

$$est_i^* := \max\{est_i , \min\{v_j , v_k\}\} , \quad (16.6.13)$$

where

$$v_j := \begin{cases} est_j + p_j & \text{if } \max\{est_j + p_j + p_i, est_i + p_i\} \leq lst_k , \\ est_k + p_k & \text{otherwise,} \end{cases}$$

$$v_k := \begin{cases} est_k + p_k & \text{if } \max\{est_k + p_k + p_i, est_i + p_i\} \le lst_j, \\ est_j + p_j & \text{otherwise.} \end{cases}$$

Proof. The assumptions imply that $(i+ii)$ is not satisfiable. From Lemma 16.6.1, we know that $est_i^* := \max\{est_i, \min\{v_1, v_2\}\}$ is the minimal earliest start time which satisfies (iii) or (iv). Further, Lemma 16.6.3 implies that there exists no smaller est_i^* satisfying $(v+vi)$, so indeed est_i^* is the correct adjustment. □

Lemma 16.6.3 leads to the consistency tests

$$\max_{u\in\{i,j,k\}, v\in\{j,k\}, u\neq v} \{lct_v - est_u\} < p_i + p_j + p_k \Rightarrow$$
$$est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k, est_j + p_j, est_k + p_k\}, \quad (16.6.14)$$

$$est_i + p_i > \max\{lst_j, lst_k\} \Rightarrow$$
$$est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k, est_j + p_j, est_k + p_k\}. \quad (16.6.15)$$

which correspond with the two different versions of the output domain consistency tests for triples of tasks (see Theorems 16.4.13 and 16.4.17). Observe that

$$LB_3(\{j,k\}) = \max\{\min\{est_j, est_k\} + p_j + p_k, est_j + p_j, est_k + p_k\}$$

is the optimal makespan if the tasks T_j and T_k are scheduled with preemption allowed. From Theorem 16.4.16, we know that the time bound adjustment $LB_3(\{j,k\})$ can be replaced with $LB_2(\{j,k\}) = \min\{est_j, est_k\} + p_j + p_k$, so that instead of (16.6.14) the following consistency test can be applied:

$$\max_{u\in\{i,j,k\}, v\in\{j,k\}, u\neq v} \{lct_v - est_u\} < p_i + p_j + p_k \Rightarrow$$
$$est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k\}. \quad (16.6.16)$$

Likewise, we can replace (16.6.15) with the equivalent consistency test

$$est_i + p_i > \max\{lst_j, lst_k\} \Rightarrow$$
$$est_i := \max\{est_i, \min\{est_j, est_k\} + p_j + p_k\}. \quad (16.6.17)$$

This follows from the fact that the 2-b-consistency tests already ensure

$$est_i \ge \max\{est_j + p_j, est_k + p_k\} \text{ if } est_i + p_i > \max\{lst_j, lst_k\}.$$

Lemma 16.6.4 derives the consistency test

$$\max_{u\in\{j,k\}} \{lct_v - est_i\} < p_i + p_j + p_k \Rightarrow est_i := \max\{est_i, \min\{v_j, v_k\}\} \quad (16.6.18)$$

which corresponds to the input negation domain consistency test for triples of tasks (see Theorem 16.4.19). Again, we can replace the time bound adjustment

$LB_6(\{j,k\}) = \min\{v_j, v_k\}$ with $LB_5(\{j,k\}) = \min\{ect_j, ect_k\}$ due to Lemma 16.4.21 which leads to the equivalent consistency test

$$\max_{u \in \{j,k\}} \{lct_v - est_i\} < p_i + p_j + p_k \Rightarrow est_i := \max\{est_i, \min\{ect_j, ect_k\}\} \quad (16.6.19)$$

This proves the assertions made in Section 16.4.3.

References

AC91 D. Applegate, W. Cook, A computational study of the job shop scheduling problem, *ORSA Journal on Computing* 3, 1991, 149-156.

Ama70 S. Amarel, On the representation of problems and goal-directed procedures for computers, in: R. Banerji, M. Mesarovic (eds.), *Theoretical Approaches to Non-Numerical Problem Solving*, Springer, Heidelberg, 1970, 179-244.

BDP96 J. Błażewicz, W. Domschke, E. Pesch, The job shop scheduling problem: conventional and new solution techniques, *Eur. J. Oper. Res.* 93, 1996, 1-33.

Bee92 P. van Beek, Reasoning about qualitative temporal information, *Artif. Intell.* 58, 1992, 297-326.

Bes94 C. Bessiere, Arc-consistency and arc-consistency again, *Artif. Intell.* 65, 1994, 179-190.

BFR99 C. Bessiere, E. C. Freuder, J.-C. Regin, Using constraint metaknowledge to reduce arc consistency computation, *Artif. Intell.* 107, 1999, 125-148.

Bib88 W. Bibel, Constraint satisfaction from a deductive viewpoint, *Artif. Intell.* 35, 1988, 401-413.

BJK94 P. Brucker, B. Jurisch, Z. Krämer, The job shop problem and immediate selection, *Ann. Oper. Res.* 50, 1994, 73-114.

BJS94 P. Brucker, B. Jurisch, B. Sievers, A fast branch and bound algorithm for the job shop scheduling problem, *Discret Appl. Math.* 49, 1994, 107-127.

BL95 P. Baptiste, C. LePape, A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling, in: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995, 136-140.

BL96 P. Baptiste, C. LePape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, in *Proceedings of the 15th Workshop of the U. K. Planning Special Interest Group*, Liverpool, 1996.

Car82 J. Carlier, The one machine sequencing problem, *Eur. J. Oper. Res.* 11, 1982, 42-47.

Che99 Y. Chen, Arc consistency revisited, *Inf. Process. Lett.* 70, 1999, 175-184.

Chr75 N. Christofides, *Graph Theory: An Algorithmic Approach*, Academic Press, London, 1975.

CL95 Y. Caseau, F. Laburthe, Disjunctive scheduling with task intervals, Technical report 95-25, Laboratoire d'Informatique de l'Ecole Normale Superieure, Paris, 1995.

Clo71 M. B. Clowes, On seeing things, *Artif. Intell.* 2, 1971, 179-185.

Coh90 J. Cohen, Constraint logic programming languages, *Commun. ACM* 33, 1990, 52-68.

Coo89 M. C. Cooper, An optimal k -consistency algorithm, *Artif. Intell.* 41, 1989, 89-95.

CP89 J. Carlier, E. Pinson, An algorithm for solving the job shop problem, *Manage. Sci.* 35, 1989, 164-176.

CP90 J. Carlier, E. Pinson, A practical use of Jackson's preemptive schedule for solving the job shop problem, *Ann. Oper. Res.* 26, 1990, 269-287.

CP94 J. Carlier, E. Pinson, Adjustments of heads and tails for the job shop problem, *Eur. J. Oper. Res.* 78, 1994, 146-161.

Dav87 E. Davis, Constraint propagation with interval labels, *Artif. Intell.* 32, 1987, 281-331.

Dew92 G. Deweß, An existence theorem for packing problems with implications for the computation of optimal machine schedules, *Optimization* 25, 1992, 261-269.

DP88 R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, *Artif. Intell.* 34, 1988, 1-38.

DPP99 U. Dorndorf, T. Phan-Huy, E. Pesch, A survey of interval capacity consistency tests for time and resource constrained scheduling, in: J. Weglarz (ed.), *Project Scheduling - Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1999, 213-238.

DPP00 U. Dorndorf, E. Pesch, T. Phan-Huy, Constraint propagation techniques for disjunctive scheduling problems, *Artif. Intell.* 122, 2000, 189-240.

DPP01 U. Dorndorf, E. Pesch, T. Phan-Huy, Solving the open shop scheduling problem, *J. Sched.* 4, 2001, 157-174.

ELT91 J. Erschler, P. Lopez, C. Thuriot, Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement, *Revue d'Intelligence Artificielle* 5, 1991, 7-32.

FN00 F. Focacci, W. Nuijten, A constraint propagation algorithm for scheduling with sequence dependent setup times, in: U. Junker, S.E. Karisch, S. Tschöke (eds.), *Proceedings of the 2nd International Workshop on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Paderborn, March 8-10, 2000, 53-55.

Fre78 E. C. Freuder, Synthesizing constraint expressions, *J. ACM* 21, 1978, 958-966.

GJ79 M. R. Garey, D. S. Johnson, *Computers and Intractability:* A *Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

HDT92 P. van Hentenryck, Y. Deville, C.-M. Teng, A generic arc consistency algorithm and its specializations, *Artif. Intell.* 57, 1992, 291-321.

Hen92 P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, 1992.

HL88 C.-C. Han, C.-H. Lee, Comments on Mohr and Henderson's path consistency algorithm, *Artif. Intell.* 36, 1988, 125-130.

HS79 R. M. Haralick, L. G. Shapiro, The consistent labelling problem: Part I, *IEEE Trans. Pattern Anal. Mach. Intell.* 1, 1979,173-184.

HS80 R. M. Haralick, L. G. Shapiro, The consistent labelling problem: Part II, *IEEE Trans. Pattern Anal. Mach. Intell.* 2, 1980, 193-203.

Huf71 D. *Z.* Huffman, Impossible objects as nonsense sentences, *Machine Intelligence* 6, 1971, 295-323.

JCC98 P. Jeavons, D. Cohen, M.C. Cooper, Constraints, consistency and closure, *Artif. Intell.* 101, 1998, 251-265.

Kum92 V. Kumar, Algorithms for constraint satisfaction problems, *AI Mag.* 13, 1992, 32-44.

LEE92 P. Lopez, J. Erschler, P. Esquirol, Ordonnancement de tâches sous contraintes: une approche énergétique, *RAIRO Automatique, Productique, Informatique Industrielle* 26, 1992, 453-481.

Lho93 O. Lhomme, Consistency techniques for numeric CSPs, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993, 232-238.

Mac77 *Z.* K. Mackworth, Consistency in networks of relations, *Artif. Intell.* 8, 1977, 99-118.

Mac92 *Z.* K. Mackworth, The logic of constraint satisfaction, *Artif. Intell.* 58, 1992, 3-20.

Mes89 P. Meseguer, Constraint satisfaction problems: an overview, *AI Commun*. 2, 1989, 3-17.

MF85 *Z.* K. Mackworth, E. C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artif. Intell.* 25, 1985, 65-74.

MH86 R. Mohr, T. C. Henderson, Arc and path consistency revisited, *Artif. Intell.* 28, 1986, 225-233.

Mon74 U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Inf. Sci.* 7, 1974, 95-132.

Moo66 R. E. Moore, *Interval Analysis*, Prentice Hall, Englewood Cliffs, 1966.

MS96 P. Martin, D. B. Shmoys, A new approach to computing optimal schedules for the job shop scheduling problem, *Proceedings of the 5th International IPCO Conference*, 1996.

MT63 J. F. Muth, G. L. Thompson (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, 1963.

NL98 W. P. M. Nuijten, C. Le Pape. Constraint-based job shop scheduling with ILOG scheduler, *J. Heuristics* 3, 1998, 271-286.

NS00 W. Nuijten, F. Sourd, New time bound adjustment techniques for shop scheduling, in: P. Brucker, S. Heitmann, J. Hurink, S. Knust (eds.), *Proceedings of the 7th International Workshop on Project Management and Scheduling*, 2000, 224-226.

Nui94 W. P. M. Nuijten, *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach,* Ph.D. thesis, Eindhoven University of Technology, 1994.

Pha96 T. Phan-Huy, *Wissensbasierte Methoden zur Optimierung von Produktionsabläufen*, Master's thesis, University of Bonn, 1996.

Pha00 T. Phan-Huy, *Constraint Propagation in Flexible Manufacturing*, Springer, 2000.

PT96 E. Pesch, U. Tetzlaff, Constraint propagation based scheduling of job shops, *INFORMS J. Comput.* 8, 1996, 144-157.

RS64 B. Roy, B. Sussman, Les problèmes d`ordonnancement avec contraintes disjonctives, Note D. S. 9, SEMA, Paris, 1964.

Sei81 R. Seidel, A new method for solving constraint satisfaction problems, *Proceedings of the 7th International Joint Conference on AI*, 1981, 338-342.

TF90 E. P. K. Tsang, N. Foster, Solution synthesis in the constraint satisfaction problem, Technical report csm-142, Department of Computer Sciences, University of Essex, Essex, 1990.

TL00 P. Torres, P. Lopez, Overview and possible extensions of shaving techniques for job-shop problems, in: U. Junker, S. E. Karisch, S. Tschöke (eds), *Proceedings of the 2nd International Workshop on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Paderborn, March 8-10, 2000, 181-186.

Tsa93 E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, Essex, 1993.

Wal72 D. L. Waltz, Generating semantic descriptions from drawings of scenes with shadows, Technical report AI-TR-271, M.I.T., 1972.

Wal75 D. L. Waltz. Understanding line drawings of scenes with shadows, in P. H. Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975, 19-91.