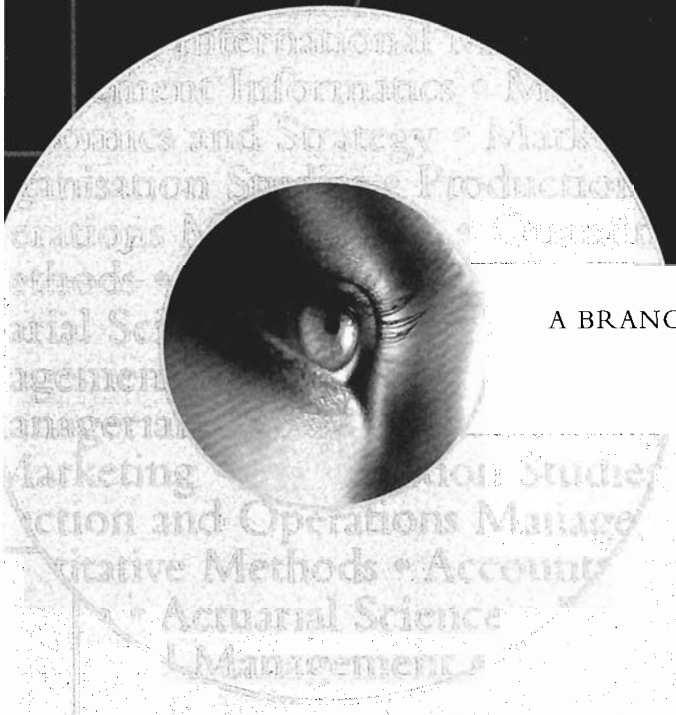

International
Management Informatics • M
Economics and Strategy • Mark
Organisation Studies • Production
Operations M • Quanti
Methods •
Actuarial Sc
Management
Managerial
Marketing
Organisation Studies
Production and Operations Manage
Quantitative Methods • Account
• Actuarial Science
Management •

A BRANCH-AND-BOUND ALGORITHM FOR STABLE SCHEDULING IN SINGLE-MACHINE PRODUCTION SYSTEMS

ROEL LEUS • WILLY HERROELEN

OR 0428

A branch-and-bound algorithm for stable scheduling in single-machine production systems

Roel Leus • Willy Herroelen
Katholieke Universiteit Leuven, Department of Applied Economics
Naamsestraat 69, 3000 Leuven, Belgium
{Roel.Leus ; Willy.Herroelen}@econ.kuleuven.ac.be

Robust scheduling aims at the construction of a schedule that is protected against uncertain events. A stable schedule is a robust schedule that will change little when variations in the input parameters arise. This paper proposes a branch-and-bound algorithm for optimally solving a single-machine scheduling problem with stability objective, when a single job is anticipated to be disrupted.

Key words: single-machine scheduling; uncertainty; robustness; branch-and-bound

1. Introduction

Manufacturing schedules are rarely executed in a 'vacuum' environment and regularly suffer disruptions from a variety of sources like resource unavailability, tardy deliveries of material or sub-assemblies, altered work content of some jobs, etc. Anticipation of uncertainty in the planning stage can be done in multiple ways. A first option is to eliminate the use of schedules altogether and construct scheduling policies that will determine dynamically which jobs to dispatch at which time instances. We refer to Part 2 of Pinedo (2002) for a survey in machine scheduling and to Stork (2001) for a project scheduling setting. Alternatively, a schedule can be constructed despite the uncertainty inherent to the scheduling environment. Such a *predictive schedule* or *pre-schedule* serves very important functions (Mehta and Uzsoy 1998). The first is to allocate resources to the different activities to optimise some measure of performance – it may be necessary to make advance bookings of key staff or equipment to guarantee their availability. The second, as also pointed out by Wu et al. (1993), is to serve as a basis for planning external activities such as material procurement, preventive maintenance and delivery of orders to external or internal customers. Pre-schedules are the starting point for communication and coordination with external entities in the company's inbound and outbound supply chain: they are the basis for agreements with suppliers and subcontractors, as well as for commitments to customers. We refer the reader to Mehta and Uzsoy (1998) and to Aytug et al. (2004) for a further discussion of the usefulness of a pre-schedule.

For our purposes, a pre-schedule with express anticipation of disruptions (which is up to a certain degree 'protected') is called *robust*. When disruptions occur during schedule execution, the pre-schedule needs to be rescheduled. Logically, robust scheduling generally incorporates assumptions about the rescheduling strategy that will be followed. If we wish to exploit the aforementioned coordination purposes of a schedule to the best possible extent, it will be desirable that execution remain as 'close' as possible to the pre-schedule. The term *stability* refers to the situation where there is little deviation between the pre-schedule and the executed schedule. The thus-defined stability concept has in the literature also been termed *solution robustness* or *predictable scheduling* and constitutes a particular form of schedule robustness. Stability can be strived for during rescheduling, and is then alternatively referred to as *minimally disruptive*, *minimal perturbation* and *minimum deviation scheduling*; for examples we refer to Akturk and Gorgulu (1999), Bean et al. (1991), Calhoun et al. (2002), Raheja and Subramaniam (2002), Rangsaritratsamee et al. (2004) and Wu et al. (1993). The option that we explore in this paper is to introduce stability already into the pre-schedule. Examples from literature are sparse, we mention Mehta and Uzsoy (1998), O'Donovan et al. (1999) and Leus (2003). Sevaux and Sörensen (2003) construct single-machine pre-schedules that are solution-robust and at the same time do not deviate much from an input *baseline* solution (for instance the basic weekly production schedule), which is a still different application of stability.

In the following section, we will introduce some notation and outline the problem we wish to solve. Section 3 presents a mathematical formulation and explains how the model is solved. Our computational experiments with the proposed algorithms are presented in Section 4 and we round off the paper with some conclusions (Section 5).

2. Notation and problem statement

Uncertainty during schedule execution is modelled by variability in job durations. In light of the difficulty of scheduling with continuous duration distributions, a number of studies have resorted to modelling duration variability by means of discrete scenarios, we refer to Daniels and Carrillo (1997), Daniels and Kouvelis (1995) and Kouvelis and Yu (1997). This will also be our choice, and we additionally evade inherent complexity due to the possible combinations of job duration realisations by optimising for the situation in which a *single* job is expected to deviate from its pre-schedule duration. For the alternative, where the job durations would be independent, the evaluation of most objective functions boils down to the generalised PERT-problem, which is particularly difficult (see Hagstrom 1988). The resulting restricted model is useful when disturbances are sparse and spread throughout time, such that the number of interactions is limited. This is especially applicable when resources are not machines but human beings, such that job durations are not mere realisations of nature but rather manageable to a certain extent. Examples in which comparable suggestions have been made are Adiri et al. (1989) (a single deterministic or stochastic breakdown), Leon et al. (1994) (one disruption on a single fallible machine in a job shop) and Mehta and Uzsoy (1998) (minimise the distance from a schedule with all jobs disrupted). In a reactive rather than proactive (robust) setting, we find similar considerations in Hall and Potts (2004) (analysis of schedule disruptions caused by the arrival of a single set of new jobs). Finally, a reasoning quite akin to ours in a graph coloring context can be found in Yáñez and Ramírez (2003), where the robustness of a coloring is measured as the probability of the coloring remaining valid after *one* random complementary edge is added to the edge set.

We assume that a set of jobs N, $|N| = n$, with deterministic durations d_i, $i \in N$, is to be scheduled on a single machine, a solution being a pre-schedule $\mathcal{S}$ that specifies starting times $s_i(\mathcal{S})$ for all jobs i. We impose a deadline ω on the pre-schedule: $s_i(\mathcal{S}) + d_i \leq \omega, \forall i \in N$. A probability of disruption p_i is associated with every job $i \in N$, which reflects the relative chance of the job suffering a perturbation in its duration. In our optimisation model, p_i is the probability that job i is the unique disrupted job, and so we assume $\sum_N p_i = 1$. We underline the fact that this single-disruption restriction is a simplification in the *model*, but that we do not impose this constraint on the *actual* job duration realisations! More in particular, any number r of activities can undergo a duration disruption by selection of r activities without replacement out of N, the probability of selection of activity i each time proportional to p_i. Our computational results (see Section 4) will show that the model is in fact quite robust to variations in the actual number of disrupted jobs. Random variable L_i denotes the increase in pre-schedule duration d_i for job i if i is disturbed. L_i is assumed discrete with probability mass function (pmf) $g_i(\cdot)$, which associates non-zero probability with positive values $l_{ik} \in \Psi_i$, where Ψ_i denotes the set of disturbance scenarios for the duration of job i; $\sum_{k \in \Psi_i} g_i(l_{ik}) = 1$ and we write g_{ik} as shorthand for $g_i(l_{ik})$; the l_{ik}-variables are indexed from small to large. Note that any continuous distribution can be approximated by choosing $|\Psi_i|$ appropriately large.

Logically, the actual starting time of job i is a random variable $S_i(\mathcal{S})$, which is dependent on the pre-schedule. Stability considerations will often make it undesirable, even impossible, to commence processing of a job earlier than its pre-scheduled starting time: job execution cannot start before auxiliary resources and tooling are freed elsewhere in the shop and before the necessary parts and materials are delivered to the processing site, and the parties responsible for these prerequisites have normally been communicated as due date the pre-schedule starting time at the initial schedule development. We model this restriction by imposing that jobs are not started earlier than foreseen, i.e. $s_i(\mathcal{S}) \leq S_i(\mathcal{S}), \forall i \in N$, which guarantees that actual production will strictly cling to the pre-schedule if all goes as planned (no disruptions). Areas of scheduling where such constraints have already been explicitly recognised are 'real-life' environments such as course scheduling, sports timetabling and railway and airline scheduling.

When the pre-schedule is implemented, disruption incident information for a particular job logically becomes available only when the job is executed. Actually, the exact timing is not even important since we reschedule simply by right-shifting the remaining jobs without re-sequencing. If we define $[i]$ to be the job that is scheduled in the i-th position, then $S_{[1]}(\mathcal{S}) = s_{[1]}(\mathcal{S})$ and $S_{[i]}(\mathcal{S}) = \max\{s_{[i]}(\mathcal{S}); S_{[i-1]}(\mathcal{S}) + D_{[i-1]}\}, i = 2, ..., n$, with D_i a stochastic variable representing the actual duration of i according to the disruption scheme described above. A non-negative cost c_i is incurred per unit-time overrun on the start time of job i, to penalise the resulting system nervousness and shop coordination difficulties (*internal* stability) as well as the delivery delay towards the customer (*external* stability). The expected weighted deviation between *actual* and *planned* job starting times (the latter corresponding with the pre-schedule) is used as stability measure for a pre-schedule $\mathcal{S}$: we minimise objective function $\sum_N c_j(ES_j(\mathcal{S}) - s_j(\mathcal{S}))$, where E is the expectation operator. For encoding reasons, we require that all values c_i and p_i be rational numbers represented by two integers, and that g_i map into the set of rational numbers. Disruption lengths l_{ik} are assumed to be integer.

The scheduling problem as set out above has been shown to be $\mathcal{NP}$-hard in the ordinary sense by Leus and Herroelen (2004), even if all $|\Psi_i| = 1$, by means of a reduction from $P2||\Sigma w_j C_j$ (whose decision problem version was proved to be ordinarily $\mathcal{NP}$-complete via reduction from KNAPSACK by Bruno et al. 1974). In fact, a proof similar to the one in Leus and Herroelen (2004) can be set up to show strong $\mathcal{NP}$-hardness by reduction

job i	1	2	3	4	5	6
p_i	0.2	0.05	0.3	0.1	0.25	0.1
$\|\Psi_i\|$	2	2	1	2	2	1
$l_{i1}(g_{i1})$	1(0.5)	1(0.7)	2(1)	2(0.5)	1(0.5)	2(1)
$l_{i2}(g_{i2})$	2(0.5)	2(0.3)	-	4(0.5)	2(0.5)	-
$p_i E_{L_i} L_i / c_i$	0.3	0.065	0.6	0.3	0.09375	0.05

Table 1: Disruption data for the example problem.

from $P||\Sigma w_j C_j$, which is said by the website on complexity results for scheduling problems maintained by Peter Brucker and Sigrid Knust to have been shown strongly $\mathcal{NP}$-hard, based on an unpublished reference of Jan-Karel Lenstra (see http://www.mathematik.uni-osnabrueck.de/research/OR/class/). It is clear that the problem has an irregular objective function, so that an optimal solution need not necessarily exist without inserted idle time and a permutation of the jobs may not suffice to produce a solution. A survey of classical scheduling problems with this characteristic is given in Kanet and Sridharan (2000); in our particular environment of variable activity durations, inserted idle time can be envisaged as buffer time to cushion the propagation of a disruption towards the (machine) successors of the disrupted job. It is important to see that the evaluation of the objective function for a given solution can be performed in polynomial time ($O(n^2 \max_{i \in N} |\Psi_i|)$), in other words, the intractability of the PERT-problem is not an issue here. Rather, the resource-allocation aspect itself seems to induce the complexity of the problem.

We illustrate the problem setting by means of a small example. Consider an instance of the described problem with $n = 6$ (number of jobs), all jobs having equal disruption probability $p_i = \left(\frac{1}{6}\right)$. Tasks indexed 5 and 6 are considered to be of high importance, the cost of delay in their starting times is $c_5 = c_6 = 4$; the other jobs $i \neq 5, 6$ have $c_i = 1$. All jobs have equal duration $d_i = 1$, and a time horizon of $\omega = 9$ time units is allotted to the set of jobs (e.g. one day's production shift length), such that we effectively have three spare units of time that can serve as buffer. Further information about the disruption scenarios of the different jobs is provided in Table 1.

Probabilities p_i of disruption reflect the relative chance of each job suffering a perturbation in its duration, with the possible perturbation lengths with their associated probabilities provided by the table. Job 1, for instance, has a relative probability of two out ten of suffering a duration disruption, and when it does undergo such disruption, this will be an increase of either one or two time units, both equally likely.

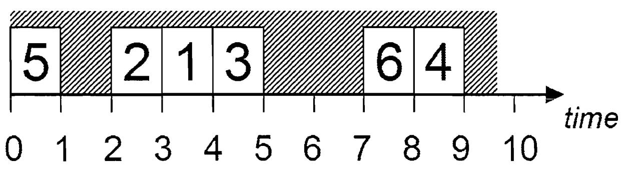


Figure 1: Optimal schedule for the example problem when $\omega = 9$.

An optimal solution to the corresponding scheduling problem (a mathematical formulation of the problem will be given in the next section) is depicted in Figure 1, the optimal objective function value is 1.005. Clearly, the available idle time is put to good use: if we reduce ω to 6 (no idle time anymore), the optimal solution has an associated cost of 4.08 for optimal job sequence 6-2-5-4-1-3. Sequence 5-2-1-3-6-4 (optimal for $\omega = 9$) corresponds with a cost of 8.455 when $\omega = 6$, whereas 6-2-5-4-1-3 achieves a cost of 1.435 when the scheduling horizon is nine time units. We point out that when the available float is zero, i.e. for the case $\omega = \sum_{i \in N} d_i$ (= 6 for the example), ordering the jobs in non-decreasing expected weighted disruption length $p_i E_{L_i} L_i / c_i$, with E_{L_i} the expectation operator with respect to L_i, leads to an optimal schedule, which is easily shown by an adjacent interchange argument as pointed out in Leus and Herroelen (2004). We will refer to this rule as the *EWDL*-rule (for *expected weighted disruption length*).

3. Model formulation and solution

In light of the discussion in the previous section on the complexity status of the problem at hand, an optimal algorithm with better than exponential time complexity is unlikely to exist, and we will devise a branch-and-bound algorithm to perform implicit enumeration of the solution space. Section 3.1 presents a general mathematical formulation of the problem to be solved. In the subsequent Sections 3.2–3.4, we expound the branch-and-bound approach. In our lower-bound computations, we efficiently apply network-flow algorithms, a point on which we elaborate in Section 3.5.

3.1. Model formulation

For convenience, we first define some decision variables that will be used throughout the remainder of this paper.

$X_{ip} = 1$ if job i is processed in position p, 0 otherwise

F_p = the distance (or buffer size) between the jobs in position p and $(p+1)$

Δ_{ijk} = the delay in the start time of job j due to a disturbance according to scenario k of job i

Clearly, inserting idle time before the job in first position or after the job in last position always leads to a dominated solution, so we only consider non-zero buffer sizes F_p for positions $p = 1, ..., n-1$. The scheduling problem under study can now be formulated as follows, with $\alpha_{ijk} = p_i g_{ik} c_j$:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{|\Psi_i|} \alpha_{ijk} \Delta_{ijk} \tag{1}$$

subject to

$$\sum_{p=1}^{n} X_{ip} = 1 \qquad i = 1, ..., n \tag{2}$$

$$\sum_{i=1}^{n} X_{ip} = 1 \qquad p = 1, ..., n \tag{3}$$

$$\Delta_{ijk} + \sum_{r=p}^{q-1} F_r \geq l_{ik}(X_{ip} + X_{jq} - 1) \tag{4}$$

$$i, j, p, q = 1, ..., n; k = 1, ..., |\Psi_i|; i \neq j; p < q$$

$$\sum_{p=1}^{n-1} F_p = \omega - \sum_{i=1}^{n} d_i \tag{5}$$

$$\text{all } \Delta_{ijk}, F_p \geq 0; \text{ all } X_{ip} \in \{0, 1\} \tag{6}$$

The objective (1) is equal to the earlier presented expression $\sum_N c_j(ES_j(\mathcal{S}) - s_j(\mathcal{S}))$, in which the expected value of the starting time delay of activity j is computed by summing the values Δ_{ijk} weighted with probability $p_i g_{ik}$. Equations (2) and (3) ensure that each position corresponds with exactly one job. Restrictions on the values Δ_{ijk} are imposed by (4) for indexes i and j that are assigned to positions p and q, respectively (the other equations are not restrictive). The corresponding delay in the start time of job j due to disruption in job i is equal to l_{ik}, the disruption length of i, minus $\sum_{r=p}^{q-1} F_r$, the buffer size in place between the positions p and q. Finally, equation (5) specifies the available total buffer space.

The foregoing model can be seen to be of the following structure:

$$\min \mathbf{a}'\mathbf{y}$$

subject to

$$\begin{aligned}\mathbf{Ax} + \mathbf{By} &\geq \mathbf{b} \\ \mathbf{y} &\geq \mathbf{0} \\ \mathbf{x} &\in \mathcal{X}\end{aligned} \tag{P}$$

with $\mathcal{X}$ the set of $\mathbf{x}$-vectors corresponding with X_{ip}-values that represent valid sequences (clearly, $|\mathcal{X}| = n!$).

An order relation is a subset of the Cartesian product $C \times C$ of its ground set C (in the context of the paper, a set of job pairs) fulfilling the requirements of reflexivity, anti-symmetry and transitivity. For a binary relation C, we can write aCb to mean that (a, b) is in C. A *complete* or *total* order relation additionally satisfies the comparability condition that either aCb or bCa for any a, b. Clearly, there is a one-to-one correspondence between each $\mathbf{x} \in \mathcal{X}$ and a total ordering of N.

3.2. General approach of the branch-and-bound algorithm

In this section, we describe the development of a branch-and-bound algorithm for solving (P). The solution space is scanned by partitioning $\mathcal{X}$ into subsets $\mathcal{X}_h$ and solving (P) for each of the restrictions $\mathbf{x} \in \mathcal{X}_h$. When $\mathcal{X}_h$ is restricted to a singleton, (P) boils down to inserting buffers into a fully specified job sequence. In this case, $\mathcal{X}_h$ induces a total ordering A_h on N: for all two jobs $i, j \in N$, $i \neq j$, either $(i, j) \in A_h$ or $(j, i) \in A_h$. During our search, we will also consider subsets $\mathcal{X}_h \subseteq \mathcal{X}$ defined by a partial ordering A_h of N and containing all $\mathbf{x}$-vectors with associated complete order $T(\mathbf{x})$ such that $A_h \subseteq T(\mathbf{x})$. We denote by $\varphi(N, A, \omega)$ the optimal objective function value of (P) when the imposed deadline is ω and the solution space is restricted to $\mathbf{x}$-vectors having $A \subseteq T(\mathbf{x})$. In other words, for any subset $\mathcal{X}_h \subseteq \mathcal{X}$, $\varphi(N, A_h, \omega)$ is the best (minimal) objective value reachable by any individual $\mathbf{x} \in \mathcal{X}_h$.

The branch-and-bound algorithm for (P) proposed in this paper proceeds as follows. From front to back of the machine, we fill one job position at a time, and each level of the search tree is associated with the filling of one position. In this way, the number of nodes at level z in case of *full* enumeration equals $n!/(n-z)!$ (=the number of permutations of z elements from n). We initialise set $J_0 = \varnothing$ and order relation $A_0 = \varnothing$. A node h at level $z(h)$ in the search tree corresponds with a subset $\mathcal{X}_h \subseteq \mathcal{X}$ and A_h defining $\mathcal{X}_h$ imposes

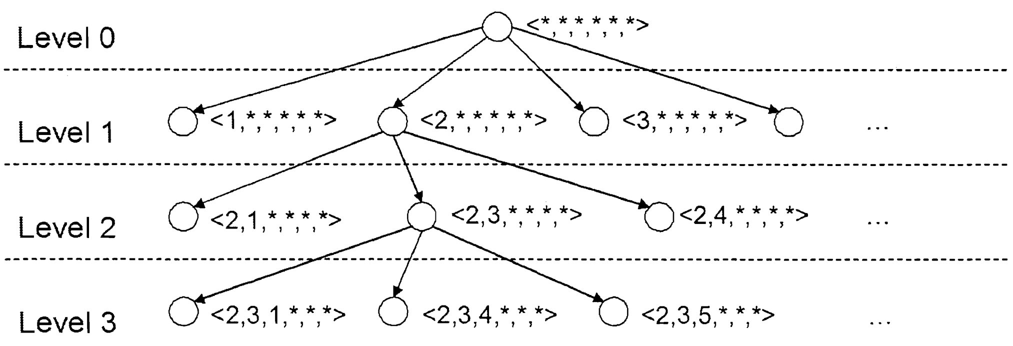


Figure 2: Illustration of the branching scheme. The corresponding set A_h is described next to each node h.

a complete order on subset $J_h \subseteq N$ of size $z(h)$. Per level in the search tree, we append one job to a partial sequence that comprises only the jobs in $J_h \subseteq N$ that have so far been added and in which the jobs are sequenced in order of addition; each node h has $n - z(h)$ child nodes; remark that $z(h) = |J_h|$. Movement to node l from node h by branching corresponds with the selection of one element $\sigma_l \in N \backslash J_h$, and we construct $J_l = J_h \cup \{\sigma_l\}$ and $A_l = A_h \cup \{(i, \sigma_l) | i \in J_h\}$, so we have $J_h = \bigcup_{m=1}^{z(h)} \{\sigma_m\}$. We can extend A_l with $\{(\sigma_l, i) | i \in N \backslash J_h\}$, but this is not used for objective function bounding (see Section 3.3). An illustration of the branching scheme is provided in Figure 2.

Nodes in the search tree are numbered in order of exploration and we traverse the tree in a *depth-first* manner (or *last-in-first-out*), since at low-indexed levels, the bounds are not tight anyway, and we can reduce the computations in a node by using information from its direct parent node more easily, which will be made clear in Section 3.5 (the latter phenomenon has been referred to as the *calculation restart* advantage, see Parker and Rardin 1988).

3.3. Bounding the objective function

It is easily seen that job starting times can be obtained from the formulation as:

$$s_i = \sum_{p=1}^{n} X_{ip} \left(\sum_{q=1}^{p-1} \left(F_q + \sum_{j=1}^{n} X_{jq} d_j \right) \right) \tag{7}$$

The precedence constraints induced by the X-values can be combined explicitly into order relation A in the following way:

$$(i, j) \in A \qquad \text{iff} \qquad \exists p, q \in \{1, ..., n\} : p < q \wedge X_{ip} X_{jq} = 1 \tag{8}$$

The pair (p, q) for which $X_{ip}X_{jq} = 1$ in (8) is logically unique. By means of (quite) some re-arranging and simplifying of the terms, we can produce the following extended formulation $(\bar{\mathrm{P}})$ for the model (1)–(6). The only difference is that according to these new constraints, the last job need not end exactly at time ω but may also finish sooner.

$$\min \sum_{(i,j)\in A} \sum_{k=1}^{|\Psi_i|} \alpha_{ijk}\Delta_{ijk} \tag{9}$$

subject to

$$(2),(3),(8),\ \text{all } X_{ip} \in \{0,1\} \tag{10}$$

$$s_j \geq s_i + d_i \qquad (i,j) \in A \tag{11}$$

$$\Delta_{ijk} \geq s_i + d_i + l_{ik} + \lambda_{ij}(A) - s_j \qquad (i,j) \in A, k \in \Psi_i \tag{12}$$

$$\omega \geq s_i + d_i \qquad i \in N \tag{13}$$

$$\text{all } \Delta_{ijk}, s_i \geq 0 \tag{14}$$

Constraint set (10) restricts the search space to all complete orders A on N. Constraints (11) are necessary to avoid concurrent scheduling of any two activities and replace equations (7): the starting time of an activity equals the starting time of its direct predecessor plus its direct predecessor's duration plus the buffer size in place between the two activities, and the buffers F_q are no longer useful in the formulation so they are eliminated and we change from equalities to $\geq$-constraints. Equations (12) determine the disruption lengths and are obtained from (4): first we add the term $\sum_{r=1}^{p-1} F_r - \sum_{r=1}^{p-1} F_r + \sum_{l=1}^{q-1}\sum_{i=1}^{n} X_{il}d_i - \sum_{l=1}^{q-1}\sum_{i=1}^{n} X_{il}d_i$ $(= 0)$ to the left hand side of (4). Next we eliminate all X-values by summing only across the necessary running indexes by means of A; the starting times s_i and s_j as defined in (7) are then readily recognised. What remains is $\lambda_{ij}(A) = \sum_{l=p}^{q-1}\sum_{i=1}^{n} X_{il}d_i$, which represents the sum of the job durations between i and j according to A if $(i,j) \in A$, otherwise 0. Constraint (5) can be re-written into $s_{[n]} + d_{[n]} = \omega$, which is then translated into constraint set (13), whence the possibility of non-zero F_n.

At this point in our exposé, we make the following observation that is important enough to warrant a lemma:

Lemma 1 *Without loss of generality, we can set all job durations equal to zero, if we accordingly subtract $\sum_{i=1}^{n} d_i$ from ω.*

Proof. Model (1)–(6) remains unchanged if the proposed change is made. ■

The adaptation to the model corresponding with this lemma is assumed to have been imposed in the remainder of this paper, but was relegated to this point in the text in order to enhance readability and generality of the foregoing. Activity starting times for a particular solution are implicit from job sequencing and buffer sizing. The need for quantities λ_{ij} is also obliterated.

We have explained in Section 3.2 that a node h in the branch-and-bound tree corresponds with a subset $\mathcal{X}_h \subseteq \mathcal{X}$ defined by a partial ordering A_h of N; from equivalence between models (P) and ($\bar{\text{P}}$), $\varphi(N, A_h, \omega)$ equals the best attainable objective value of ($\bar{\text{P}}$) with extra constraint $A_h \subseteq A$. A lower bound $\varphi^\circ(N, A_h, \omega)$ for $\varphi(N, A_h, \omega)$ is obtained by replacing constraint set (10) together with $A_h \subseteq A$ by constraint $A = A_h$. For the resulting relaxation, if either i or j is in $N \backslash J_h$ then the corresponding $\Delta_{ijk} = 0$. Consequently, an optimal solution to this relaxation will have the same objective value as an optimum for ($\bar{\text{P}}$) with job set J_h, which shows that $\varphi^\circ(N, A_h, \omega) = \varphi(J_h, A_h, \omega)$. We also note that $\varphi^\circ(J, A, \omega) = \varphi(J, A, \omega)$ when A is a complete order on J, which is the case when $J = J_h$ and $A = A_h$. In leaf nodes h of the search tree (with $z(h) = n$ and $J_h = N$), $\mathcal{X}_h$ is restricted to a singleton and $\varphi^\circ(J_h, A_h, \omega)$ is the exact objective function value of the individual leaf node solution. We will discuss how function φ° is computed in Section 3.5.

Next, we focus on incorporating the expected cost of disruption of the jobs in $N \backslash J_h$. The expected disruption of a job $j \in N \backslash J_h$ by $i \in J_h$ is lower bounded by the quantity $p_i \sum_{k \in \Psi_i} g_{ik} \min\{0; l_{ik} - \omega\}$, since $N \backslash J_h$ will be appended *after* the chain of jobs J_h and no more than ω time units can be inserted between i and j to cushion disruption of i; this leads to quantity $q(\omega, h) = \sum_{(i,j) \in (J_h \times (N \backslash J_h))} p_i c_j \sum_{k \in \Psi_i} g_{ik} \min\{0; l_{ik} - \omega\}$. The lower bound $\varphi^\circ(N, A_h, \omega) + q(\omega, h)$ for $\varphi(N, A_h, \omega)$ is referred to as LB_0.

Because set J_h is entirely executed before $N \backslash J_h$, any feasible solution to (P) 'assigns' float quantity f $(0 \leq f \leq \omega)$ to $N \backslash J_h$ (to be inserted between $N \backslash J_h$-jobs) and $(\omega - f)$ is available for J_h, if we neglect float value $F_{[z(h)]}$. Therefore, in any search node h,

$$\varphi(N, A_h, \omega) \geq q(\omega, h) + \min_{0 \leq f \leq \omega} \{\varphi(J_h, A_h, \omega - f) + lb_x(f, h)\} = LB_x(\omega, h), \tag{15}$$

with $lb_x(f, h)$ a lower bound on $\varphi(N \backslash J_h, \varnothing, f)$, the expected cost of disruption of $N \backslash J_h$-jobs by other jobs in $N \backslash J_h$ (and x an integer index above zero). Two such bounds lb_x are considered. The first bound $lb_1(f, h)$ exploits the fact that scheduling with zero float is polynomially solvable. We create an auxiliary problem in which we set each disruption length in scenario k of each activity i equal to $\max\{0; l_{ik} - f\}$ and the deadline is 0. $lb_1(f, h)$ does indeed constitute a lower bound because the available float is *re*-used in its entirety to cushion disruptions in each individual activity duration; the scheduling problem is solved by the *EWDL*-rule. A different bounding approach is based on the following insight: by Jensen's Inequality, if we replace all disruption scenarios $k \in \Psi_i$ of the jobs $i \in N \backslash J_h$ by one single disruption with length $E_{L_i} L_i$, the resulting objective function is a lower bound to that of the original problem. Next, replacing all cost coefficients c_i by c_{i^*} with i^* the job in $N \backslash J_h$ with lowest cost, and likewise taking for all jobs the same lowest probability and disruption length, does not increase the objective value. For the resulting set of $|N \backslash J_h|$ *identical* jobs, sequencing is no longer needed and optimal starting times can be obtained by means of network-flow techniques (see again Section 3.5). We call the resulting bound lb_2.

For LB_2, a local minimum suffices for minimisation in f since the expression to be minimised is convex, which follows from convexity of $\varphi()$ when a full order is specified, and from convexity of lb_2, and the sum of two convex functions is also convex. The former convexity result derives from sensitivity analysis in linear programs, more specifically from the global dependence of the objective on the right-hand side vector (we obtain that $\varphi()$ is convex in $(\omega - f)$ for $f \in [0; \omega]$, and therefore also in f on the same interval), and lb_2 is also the output of a linear program. Unfortunately, lb_1 and thereby also LB_1 is not convex (for a counterexample, see Appendix A). LB_1 is computed by consideration of all discrete values f in $[0; \omega]$, while Golden Section Search can be applied for determining LB_2, in which we examine only the discrete values for f. The function to be minimised need not be unimodal, which is normally a condition for applicability of Golden Section Search, because it is convex.

It is clear that the determination of LB_1 and LB_2 may both require a significant amount of computational effort. We have therefore also implemented 'simpler' lower bounds $SLB_x = q(\omega, h) + \varphi(J_h, A_h, \omega) + lb_x(\omega, h)$, $x = 1, 2$, in which both terms in the expression to be minimised in (15) receive the maximum float ω. The SLB_x-bounds are logically never tighter than their LB_x-counterparts due to the monotonicity of φ and lb_x in f.

3.4. Further algorithmic details

In this section, we discuss the topics of dominance rules, intermediary feasible solutions, the choice of the order of exploration of branching alternatives, and pre-processing.

3.4.1. Dominance rules

A pairwise interchange argument shows that any two *consecutive* jobs i, j in an optimal solution either are in $EWDL$-order or have non-zero buffer between their positions. More in particular, when we schedule i *immediately* before j, it should hold that either $p_i c_j E_{L_j} L_j \leq p_j c_i E_{L_i} L_i$, or else a buffer of size at least 1 should be inserted between the two positions, otherwise the solution is dominated. We can restrict our search to integral buffer sizes (hence, 'non-zero' leads to the '≥ 1') since an optimal solution exists with integral starting times, which follows from our discussion in Section 3.5. This additional constraint can be explicitly imposed on the starting times of the jobs in J_h. When the cumulative minimal buffer sizes exceed ω, the current search node can be fathomed; this test is performed implicitly by the flow computations in Section 3.5. In any node h of the search tree, we let δ^h_{ij} denote the minimal distance between i and j. This gives rise to a starting time constraint in the form of (16) to replace (11) – normally with $d_i = 0$:

$$s_i + d_i + \delta^h_{ij} \leq s_j \qquad (i,j) \in A \tag{16}$$

The cumulative δ-values can be subtracted from the float f that is available for jobs $N \backslash J_h$ in lower-bound computations.

Other minimal distances can be computed between the starting times of pairs of jobs in the partial order, based on the current incumbent and lower bounds, but this has not been implemented. Such minimal distances can be dealt with in exactly the same way. Construction of the transitive closure of the resulting distance matrix as a means of constraint propagation is possible, but these transitive constraints are immediately imposed anyway by the flow model of Section 3.5.

3.4.2. Intermediary feasible solutions

Intermediary feasible solutions are constructed very easily, since any linear extension of A_h in node h is allowable; each solution yields a global upper bound. Buffer insertion is then still an issue, however. This can be performed in polynomial time (see Section 3.5), but is nonetheless costly in terms of CPU-time. We therefore resort to (a slightly adapted version of) the heuristic *ADFF* (*activity-dependent float factor*, proposed in Herroelen and Leus 2004), which simply produces activity starting times in closed-form expression rather than depend on an optimisation run, and performs better than other 'buffer insertion' heuristics.

The algorithm proceeds as follows. For a full order A on N that is input to the algorithm, the starting time of an activity i is the integer nearest to $\delta_i(A)\omega$, with

$$\delta_i(A) = \frac{\sum_{\substack{(j,k)\in A:\\ (k,i)\in A \vee k=i}} p_j E_{L_j} L_j c_k}{\sum_{\substack{(j,k)\in A:\\ (k,i)\in A \vee k=i}} p_j E_{L_j} L_j c_k + \sum_{\substack{(j,k)\in A:\\ (i,j)\in A \vee i=j}} p_j E_{L_j} L_j c_k}$$

We see that $\delta_i(A) \leq \delta_j(A)$ if $(i,j) \in A$, such that $s_i(ADFF) \leq s_j(ADFF)$, and we also have $\omega \geq s_i(ADFF)$ for every $i \in N$ since $\delta_i \in [0;1]$, so the resulting schedule is feasible. In $ADFF$, the starting time of activity i equals its earliest possible starting time 0 augmented with fraction δ_i of the available float, where δ_i attempts to measure what proportion of cost *dependent on the position of* i is related to activity pairs before i in A. During the search, we maintain an n-vector with an arbitrary permutation of the job indexes (initialised with index i in the i-th position), which is continuously updated to be compatible with all branching decisions leading to the current search node. Each time an update is needed, we re-run $ADFF$ based on the new corresponding full order on N.

3.4.3. Order of exploration

Because of the fact that the lower-bound computations are intimately tied with the incremental construction of solutions (see Section 3.5), it would be difficult to use them as the basis for determining the order of exploration of the child nodes of a node in the search tree, since the bounds would then need to be computed for *all* branching alternatives before one of the alternatives is implemented. We therefore order the candidate jobs in decreasing order of a *pseudo-cost* of insertion, which is an estimate of the true cost, but no bound. The role of this pseudo-cost is in guiding heuristic decisions in the algorithm, not in generating incumbent solutions or in proving fathomability (Parker and Rardin 1988). In our implementation, we simply scan the branching alternatives in $EWDL$-order.

3.4.4. Pre-processing

It is clear that activities with zero cost coefficient can be sequenced last: this is always a dominant decision. In fact, those activities can be removed from the problem description in a pre-processing phase without impact on the objective function. The same goes for activities i with zero $p_i E_{L_i} L_i$. Additionally, pre-processing can reduce the size of the scheduling instance, especially for small ω, in the following way. We know that $ES_j =$

$\sum_{\substack{i\in N\\ i\neq j}} p_i \sum_{k\in\Psi_i} g_{ik}\Delta_{ijk}$. If we denote by Ψ_i^1 the set of disruption scenarios of i with $l_{ik} < \omega$ and $\Psi_i^2 = \Psi_i \backslash \Psi_i^1$ then $ES_j = \sum_{\substack{i\in N\\ i\neq j}} p_i \sum_{k\in\Psi_i^1} g_{ik} \max\{0; l_{ik} - SB_{ij}\} + \sum_{\substack{i\in N\\ i\neq j}} p_i \sum_{k\in\Psi_i^2} g_{ik}(l_{ik} - SB_{ij})$, with SB the sum of the buffers in place. We can see that all scenarios in Ψ_i^2 can be replaced by one disruption scenario k^* with $l_{ik^*} = \sum_{k\in\Psi_i^2} g_{ik}l_{ik} / \sum_{k\in\Psi_i^2} g_{ik}$ and $g_{ik^*} = \sum_{k\in\Psi_i^2} g_{ik}$ without influence on the objective function. Unfortunately, we work with integer data, so in order for this rule to be implemented, the time horizon will probably need to be discretised more finely. Computationally, this poses no problem, but the dominance rule in Section 3.4.1, which imposes unit-time differences between job starting times, loses much of its value. Therefore, this pre-processing rule is not implemented.

We have already explained that when $\omega = 0$, (P) can be solved in polynomial time. Similarly, an optimal polynomial-time algorithm exists when $\omega \geq \sum_{i\in N} l_{i,|\Psi_i|} - l_{\max}$, with $l_{\max} = \max_{i\in N} l_{i,|\Psi_i|}$: schedule one activity $\arg\max_{i\in N} l_{i,|\Psi_i|}$ last, the other elements of N in arbitrary order, and insert a buffer of size $l_{i,|\Psi_i|}$ after each activity i but the last. Other isolated special cases might also allow for a dedicated polynomial-time solution procedure, but in general, for intermediary choices of ω, the required computational effort cannot be guaranteed to be polynomial. We empirically examine the running times of our algorithm as a function of ω in Section 4.

3.5. Network flows

Herroelen and Leus (2004) have examined how the scheduling of activities with a partial order and *without* resource constraints can be performed in the duration-disruption setting outlined in the foregoing sections; it turns out that this can be achieved by the solution of a linear program, the dual of which is a minimum-cost network-flow problem (MCNFP). In particular therefore, the problem can be solved in polynomial time. We will explain how their solution method is invoked to compute $\varphi(J_h, A_h, \omega)$ with A_h a full order on J_h. We first write out the underlying model, in which J_h is augmented with a dummy start node 0 and dummy end node $(n+1)$, both with zero cost and zero disruption probability, which come first and last in A_h, respectively. The model focuses on the relative position of the jobs in time rather than on absolute values of starting times, which is reflected in the absence of sign constraints for the s-variables.

$$\min \sum_{(i,j)\in A_h} \sum_{k=1}^{|\Psi_i|} \alpha_{ijk}\Delta_{ijk} \tag{17}$$

subject to

$$s_j - s_i \geq \delta^h_{ij} \qquad (i,j) \in A_h \tag{18}$$

$$\Delta_{ijk} + s_j - s_i \geq l_{ik} \qquad (i,j) \in A_h, k \in \Psi_i \tag{19}$$

$$s_0 - s_{n+1} \geq -\omega \tag{20}$$

$$\text{all } \Delta_{ijk} \geq 0; \text{ all } s_i \text{ unrestricted in sign} \tag{21}$$

If we assign non-negative multipliers x_{ij}, y_{ijk} and v to the constraints (18), (19) and (20), respectively, the dual of the foregoing model can be written as follows:

$$\max \sum_{(i,j)\in A_h} \delta^h_{ij} x_{ij} + \sum_{\substack{(i,j)\in A_h \\ k\in\Psi_i}} l_{ik} y_{ijk} - \omega v \tag{22}$$

subject to

$$\sum_{(i,j)\in A_h} x_{ij} - \sum_{(j,i)\in A_h} x_{ji} + \sum_{\substack{(i,j)\in A_h \\ k\in\Psi_i}} y_{ijk} - \sum_{\substack{(j,i)\in A_h \\ k\in\Psi_j}} y_{jik} = \begin{cases} 0 & i \in J_h, i \neq 0, n+1 \\ v & i = 0 \\ -v & i = n+1 \end{cases} \tag{23}$$

$$0 \leq y_{ijk} \leq \alpha_{ijk};\ 0 \leq x_{ij} \qquad (i,j) \in A_h, k \in \Psi_i \tag{24}$$

This is a MCNFP with node set J_h and arc set A_h augmented with return arc $(n+1, 0)$. Each arc $(i,j) \in A_h$ is actually a multi-arc, representing $|\Psi_i| + 1$ individual arcs with flow quantities x_{ij} and y_{ij1} to $y_{ij|\Psi_i|}$; x_{ij} has the lowest profit $\delta^h_{ij} = 0$ or 1 and is uncapacitated, while y_{ijk} has profit coefficient l_{ik} and flow capacity α_{ijk}.

In every node of the search tree, we could solve stand-alone MCNFPs to produce lower bounds, but we will approach this issue more efficiently: we maintain a flow network in which a flow is preserved at all times that is feasible for the current search node, such that *good* starting solutions are immediately available for step-wise primal algorithms, which proceed to optimal solutions through a direct, constructive sequence of improving feasible solutions. In our application, optimal solution of the resulting MCNFP is obtained by means of the strongly polynomial minimum-mean cycle-cancelling algorithm (Ahuja et al. 1993), in which the successive negative-cost augmenting directed cycles in the residual network are identified by the algorithm of Karp (1978) as the negative cycles with minimum mean cost (the mean cost of a cycle is its cost divided by the number of arcs it contains). Remark that the residual network is always strongly connected, given the uncapacitated x-arcs and return flow v, such that Karp's algorithm is easily implemented. In the search for a negative-cost augmenting cycle and in the longest-path computations, we can exploit the fact that from the multi-arc between any two nodes, maximum two individual arcs need actually be considered, namely the unsaturated one with maximal benefit (as forward arc) and the flow-carrying arc with minimal benefit (as backward arc); these two may be identical. Node 0 is chosen as the source for Karp's algorithm.

Intuitive insights such as those cited in Dasdan and Gupta (1998) for enhancing algorithmic efficiency have been tested but are of little value because of the density of the network. When the capacity of flow-bearing arcs is re-set to zero, for instance due to backtracking, the arc flows are removed by means of one or more augmenting cycles that contain the arcs in question as backward arcs; these augmenting cycles are identified by a (strongly polynomial) shortest augmenting-path algorithm. If the MCNFP is unbounded, the primal model is infeasible because the cumulative minimal starting-time differences δ_{ij}^h exceed ω, in which case we can fathom the current search node and backtrack (the infinite-capacity positive-gain augmenting cycle C is composed of x-arcs and v, with $\sum_{\substack{(i,j)\in C \\ \neq(n+1,0)}} \delta_{ij}^h > \omega$). Otherwise, once an optimal MCNFP-solution is found, an optimal solution to model (17)–(21) is constructed by exploiting complementary-slackness conditions for linear programs. The following cases can be distinguished:

1. $y_{ijk} = 0$. Since $\Delta_{ijk} = 0$ (complementary slackness), $s_j \geq s_i + l_{ik}$.

2. $0 < y_{ijk} < \alpha_{ijk}$. This leads to $\Delta_{ijk} + s_j - s_i = l_{ik}$ (complementary slackness) and $\Delta_{ijk} = 0$ (for the same reason), so $s_j = s_i + l_{ik}$.

3. $y_{ijk} = \alpha_{ijk}$. In this case, $\Delta_{ijk} + s_j - s_i = l_{ik}$, and since $\Delta_{ijk} \geq 0$, we obtain that $s_i \geq s_j - l_{ik}$.

For the x-arcs, we have

1. $x_{ij} = 0$. This gives $s_j \geq s_i + \delta_{ij}^h$.

2. $x_{ij} > 0$. This yields $s_j = s_i + \delta_{ij}^h$.

Using these observations, we can find the solution of the primal problem by solving a longest-path problem in the residual network (which may have negative arc lengths), where arcs lengths are equal to the minimum timelags between the job starting times – remark that the longest path from 0 to i *minimises* s_i subject to the equality and inequality constraints. For arcs in the residual network corresponding with forward arcs in the original network, only cases 1 and 2 are relevant, for backward arcs, only 2 and 3. Without loss of better solutions, we choose $s_0 = 0$ and $s_{n+1} = \omega$. The remaining starting times are well defined because the residual network does not contain a positive cycle, from optimality of the MCNFP-solution. Also, at most one arc corresponding with each multi-arc carries flow at a value strictly between its lower and upper bound, because of the structure of the profit coefficients, which allows to easily identify the predecessor disruption scenario up to which jobs are protected. The longest-path problem is solved using an adaptation of the FIFO label-correcting algorithm: from s_0 and s_{n+1}, we can obtain permanent starting times for intermediary jobs i if equality restrictions relate s_i to other permanent starting times (while in principle, for label-correcting algorithms such as the FIFO algorithm, all labels are temporary until termination of the algorithm, see Ahuja et al. 1993).

4. Computational experiments

In this section, we discuss the experimental setup of our computational experiments (Section 4.1), we provide some figures to illustrate the computational efficiency of our branch-and-bound algorithm (Section 4.2), and we compare the optimal solutions to our model with other scheduling approaches with respect to protection against uncertainty in job processing times (Section 4.3).

4.1. Experimental setup

To examine the performance of the branch-and-bound algorithm presented in Sections 3.2–3.5 and the underlying model of Section 3.1, a series of computational experiments using randomly generated test problems has been conducted. For various values of n, we have generate a dataset of 25 problems. For each activity i in each instance of these datasets, the disturbance length L_i is a discrete random variable for which g_i is a discretised version of the continuous linearly decreasing pdf $h_i(x) = 2(1/I_i - x/I_i^2)$, for which the intercept I_i with the abscissa is a realisation of a discrete uniform random variable with support $[2; 25]$. Scenarios $k \in \Psi_i$ are determined as follows: l_{i1} is randomly selected from the discrete values in $[1; \min\{4, I_i - 1\}]$ and additional scenarios $l_{ik} = l_{i,k-1} + 5$ are added while $l_{ik} \leq I_i - 1$; each $g_{ik} = h_i(l_{ik})$. For each job, a value q_i is selected from the continuous domain $[1; 8]$ and these values are then normalised to probabilities p_i. Cost coefficients c_i are integer values randomly selected from $[1;4]$. The schedule deadline ω is in most cases determined as the upper integer of a fraction ω_0 of the average disruption length $E_{L_i}L_i$, further averaged (with equal weights) across all jobs i. For the example problem of Section 2, for instance, the average disruption length is $(1.5+1.3+2+3+1.5+2)/6 = 1.8833$, such that $\omega = 3$ corresponds with $\omega_0 \in [1.062+\epsilon; 1.593]$, with ϵ a small number.

A description of the implementation of our algorithm is given in Appendix B. Our implementation takes all integer inputs. Since probabilities p_i and pmf g_i may have fractional values, the primal objective-function coefficients are multiplied by factor 10,000 and rounded to the lower integer. Our coding was performed in C, using the Microsoft Visual C++ 6.0 programming environment, and the experiments were run on a Dell Latitude D800 portable computer with Pentium M processor with 1,400 MHz clock speed and 512 MB RAM, equipped with the Windows XP operating system.

4.2. Computational efficiency

For the dataset with eight jobs per scheduling problem, we present successive improvements in the efficiency of our branch-and-bound (B&B) algorithm in Table 2. The table indicates the average percentage of number of nodes visited and of CPU-time when compared with the final version of the algorithm (setting "(5)"), which makes use of the dominance rule and the simple lower bounds SLB_1 and SLB_2, but not of the generation of intermediary feasible solutions, nor of the more involved lower bounds LB_1 and LB_2.

We notice that the amount of float ω is a key determinant for the value of the algorithmic enhancements that we apply to the base setting (1). Intermediate feasible solutions turn out to be completely useless: the number of search nodes is never even slightly reduced compared with the reference setting (5). For $\omega = 1$, considerable running-time improvements are achieved by SLB_1, SLB_2 and the dominance rule, but LB_1 and LB_2 are not able to further reduce the search space significantly. For slightly higher ω ($\omega_0 = 0.5$), the reference setting (5) is still among the best, although inclusion of LB_1 allows to gain on running time by means of the reduction of the number of nodes in the search tree by about 18%; the gain in CPU-time is less then proportionate, however. If we further increase ω, both LB_1 and LB_2 cut away a part of the search tree in comparison with case (5) but this benefit is more than offset by the incremental computational effort required by these bounds, such that in total their incorporation has a strongly disadvantageous effect on the CPU-time. When $\omega_0 = 2.5$, we observe that the algorithmic enhancements that were useful for small float values are

	$\omega = 1$		ω_0= 0.5	
	CPU	nodes	CPU	nodes
(1) = branching + LB_0	579.59%	551.02%	165.75%	178.40%
(2) = (1) + SLB_1	455.97%	415.99%	146.62%	155.59%
(3) = (1) + SLB_2	554.84%	509.28%	163.62%	171.36%
(4) = (1) + dominance rule	172.07%	195.29%	144.93%	152.44%
(5) = (1) + (2) + (3) + (4)	100.00%	100.00%	100.00%	100.00%
(6) = (5) + intermed. solutions	103.10%	100.00%	102.56%	100.00%
(7) = (5) + LB_1	102.07%	97.93%	96.90%	82.87%
(8) = (5) + LB_2	111.06%	100.00%	125.71%	99.79%

	ω_0= 1.5		ω_0= 2.5	
	CPU	nodes	CPU	nodes
(1) = branching + LB_0	102.17%	107.09%	98.69%	101.23%
(2) = (1) + SLB_1	101.29%	105.82%	98.77%	101.16%
(3) = (1) + SLB_2	106.97%	101.18%	99.95%	101.23%
(4) = (1) + dominance rule	102.40%	104.01%	99.12%	100.32%
(5) = (1) + (2) + (3) + (4)	100.00%	100.00%	100.00%	100.00%
(6) = (5) + intermed. solutions	102.24%	100.00%	102.35%	100.00%
(7) = (5) + LB_1	166.29%	71.93%	268.19%	73.53%
(8) = (5) + LB_2	234.84%	90.65%	397.51%	88.16%

Table 2: Successive improvements in the branch-and-bound algorithm.

not valuable anymore, and increase rather than decrease the computational effort. In the eight-job dataset, $\omega_0 = 2.5$ corresponds with a value for ω between 9 and 16, with an average of just below 12.

We have passed the IP-formulation (1)–(6) to the IP-solver Lindo (Industrial Lindo/PC release 6.01 (1997); the associated dynamic link library (dll) is called from our C-code). A comparison of the running times of the solver ("IP") with those of our algorithm ("B&B") in the reference setting (5) is provided in Table 3. The trends are obvious: we are able to produce optimal solutions to model (P) in considerably less computation time. We also notice that the computational effort required to produce optimal solutions goes up when ω increases.

We elaborate on this behaviour in Figure 3 for larger values of ω_0, where case $\omega_0 = 0$ refers to $\omega = 1$; in the same graph, we also provide an indication of the evolution of the average number of nodes in the search tree as well as of the optimal objective function value. We observe that the computational effort is largest for ω_0 ranging from 2 tot 4 and then decreases with increasing ω_0. The number of nodes in the search tree, on the other hand, takes on a much more moderate descent from that point onwards. One possible explanation

	$\omega = 1$		ω_0= 0.5		ω_0= 1.5		ω_0= 2.5	
n	B&B	IP	B&B	IP	B&B	IP	B&B	IP
4	0.00s	0.22s	0.00s	0.21s	0.00s	0.22s	0.00s	0.20s
6	0.02s	57.16s	0.03s	1m 6s	0.05s	1m 17s	0.05s	1m 21s
8	0.63s	>20m	2.34s	>20m	4.30s	>20m	5.19s	>20m
10	16.51s	-	2m 56s	-	8m 44s	-	12m 26s	-
12	4m 28s	-	>20m	-	>20m	-	>20m	-
14	>20m	-	-	-	-	-	-	-

Table 3: Comparison of the average CPU-times for the B&B-algorithm and the Lindo-solver, for various sizes of n. The settings without entry were not run to termination because of excessive computation time (well over 20 minutes).

for this phenomenon is that the MCNFP-computations probably consume less time, because the objective function to be reached is simply lower. We also notice that a significant stability gain can be achieved with moderate float values: the objective function steeply falls in the left part of the graph, and then evens out.

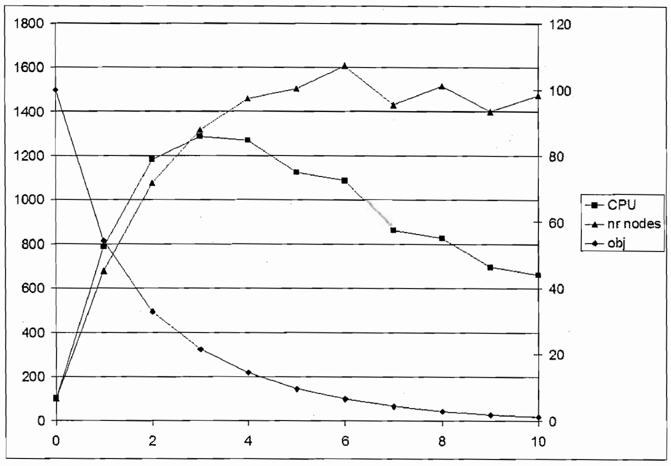


Figure 3: The evolution of the number of nodes and the CPU-time (left ordinate) and the optimal objective function (right ordinate) in function of ω_0, expressed in percentage points compared with case $\omega = 1$.

4.3. Objective function comparison with heuristics

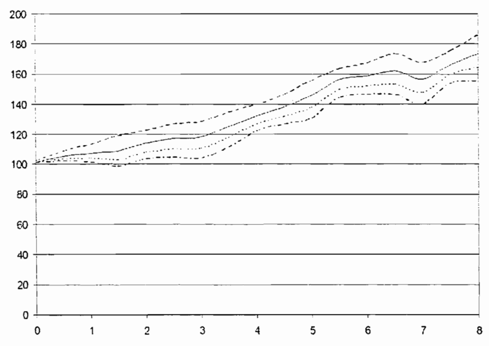
In order to examine the performance on the stability objective $\sum_N c_j(ES_j(\mathcal{S}) - s_j(\mathcal{S}))$ of the proposed one-disruption model, we will compare the output of the model with a number of different scheduling approaches: a full order on N is determined (1) as the $EWDL$-order

("E") and (2) randomly (in increasing order of job index, "I"). Afterwards, the jobs are scheduled subject to this full order, (1) by means of the *ADFF*-heuristic ("A") and (2) using the network-flow techniques of Section 3.5 ("N"). This results in four heuristics *HEA*, *HEN*, *HIA* and *HIN* (in which the second and third letter identify the sequencing and the scheduling method applied, respectively). Compared with the B&B-algorithm, the running times are negligible for all four the heuristics.

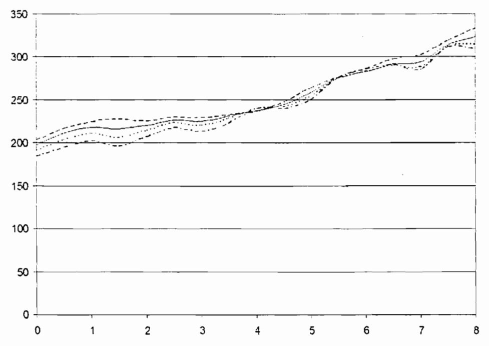
Evaluation of the stability of each schedule takes place in the following way: for producing a particular realisation of job disruption lengths, we select a pre-specified number r of jobs without replacement out of N, with probability of selection of job i each time proportional to p_i. For each thus-selected job i, one disruption length l_{ik} is chosen by picking exactly one scenario out of Ψ_i, where scenario k obviously has probability g_{ik} of being picked. The *actual* job starting times corresponding with a disruption realisation are obtained by starting each job at the maximum of the finishing time of its immediate machine predecessor and of its own pre-scheduled starting time. The weighted deviation for the corresponding realisation is then easily computed. Per scheduling instance and corresponding schedule, we estimate the *expected* weighted deviation by averaging the objective of 50,000 runs. All results in this section pertain to the case $n = 8$.

Figure 4 summarises the results of our comparison with the benchmark heuristics as a function of ω_0; $\omega_0 = 0$ again corresponds with $\omega = 1$. We see that model (P) is robust to deviations from the one-disruption assumption ($r = 1$): even when the duration of half of the activities is perturbed ($r = 4$), the schedules still strongly outperforms all heuristics, especially for large float values.

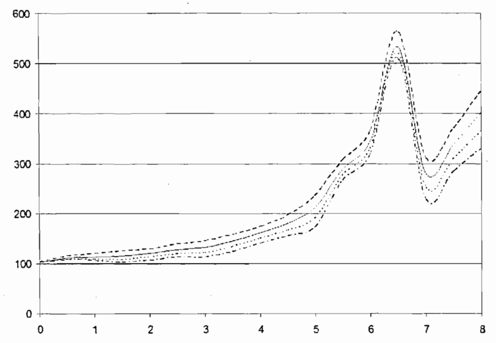
For low ω-values, the sequencing approach is the key performance determinant: *HEN* and *HEA* cross the ordinate at slightly over 100% (remember that the *EWDL*-rule is optimal for $\omega = 0$!), versus some 200% for *HIN* and *HIA*. The peak around 6.5 for *HEA* and *HIA* can be explained as follows: the optimum of (P) reaches 0 for one instance, and so we have replaced the corresponding percentage difference by 1. In the ω_0-value range up to $\omega_0 = 7$, we see that the percentage differences are rising sharply because the reference becomes lower and lower. The difference goes to infinity at 7, and the influence of the instance is 'neutralised' for this and larger ω_0. If we were to continue the abscissa beyond $\omega_0 = 8$, the same phenomenon would occur again for other instances. Since the MCNFP-method for scheduling is also able to reach this zero objective value, the same peak does not arise for *HEN* and *HIN*. These graphs suffice to show that large stability differences can come up, which would be even



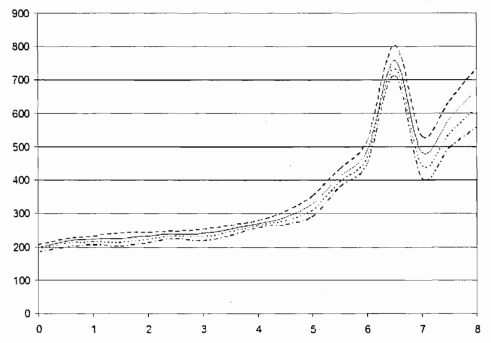
(a) Results for *HEN*



(b) Results for *HIN*



(c) Results for *HEA*



(d) Results for *HIA*

Figure 4: Comparison with heuristics: the objective function resulting from simulation is expressed in percentage points compared with the output of model (P); ω_0 is on the abscissa. The four curves (highest to lowest) correspond with $r = 1, 2, 3$ and 4, respectively.

more so if we were to compare with 'active' schedules (schedules without idle time), which completely disregard the available float time.

The results depicted in Figure 4 are based on a simulation that draws the disruption lengths form the discrete input scenarios, as explained higher in this section. There is no guarantee, however, that actual disruptions in the project under study will take on the exact same values that were input to the model, which may either stem from a discretisation of a continuous pdf (as was done here) or be the collection of past experience on similar jobs. We evaluate the robustness of our model to deviations from the input scenarios by sampling disruption lengths from the continuous function that was the basis for the selection of the input scenarios – and which in a practical setting is generally *unknown*, such that discrete approximation is indeed a good alternative. More concretely, the disruption length of job i is now a random variable $L_i = I_i(1 - \sqrt{U_i})$, with U_i a continuous random variable on domain $[0; 1]$. For these settings, we have made a comparison between our model and *HEN*, which

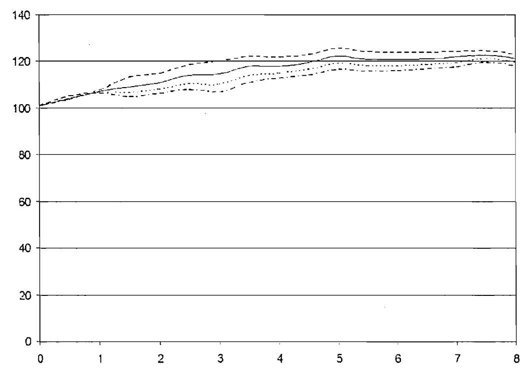


Figure 5: The results of the sampling from continuous distributions for *HEN*. The graph is constructed in a similar way as those in Figure 4.

is the best of the proposed heuristics. This yields the graph represented as Figure 5. We conclude that, although the differences are smaller, the optimal one-disruption model still performs significantly better than *HEN* for all r-values.

5. Summary and conclusions

The *stability* objective is a rather new topic in the field of scheduling under uncertainty. This paper has examined the development of a stable one-machine schedule, in which small changes due to activity duration fluctuations have only a local effect and do not propagate throughout the scheduling horizon. Deterministic schedules are proposed with explicitly inserted idle time serving as protective buffer time. A mathematical-programming model was presented to minimise the expected weighted deviation in starting times of the jobs when *exactly one* job is anticipated to suffer a deviation from its pre-schedule duration. The model was solved by means of a dedicated branch-and-bound procedure.

We conclude with the following statements.

1. By means of the one-disruption restriction, our model is more easily solvable than the case with independent activity duration distributions. This approach is not conventional and may give rise to polemic, in fact, one could argue that this line of reasoning that 'problems are caused locally, and do not interact with each other' is completely unsatisfactory. Nevertheless, the computational results that we obtain are encouraging, in that the one-disruption model produces well protected schedules for a wide range of the actual number of disrupted activities. It is our opinion that this result

justifies further examination of this pragmatic approach to dealing with uncertainty, in which only the main effects of the separate disruption of each of the n activities are considered, rather than all 2^n possible combinations of disruptions. Mere buffer insertion for a full order on the task set with independent durations already appears to constitute a formidable task.

2. As was to be expected, the computational performance of our algorithm as well as the achieved stability crucially depend on the amount of buffer time that is available to be inserted into the schedule. With respect to computation time, this is mostly because the strength of the lower bounds and the value of the dominance rule decrease with increasing float. At the same time, the number of possible solutions also dramatically increases: there is a combinatorial explosion not unlike the impact of increasing the number of buffer spaces available for buffer allocation in production lines (see e.g. Lutz et al. 1998, and Papadopoulos and Vidalis 2001). The network-flow techniques that we apply largely eliminate this latter problem, however.

3. Uncertainty is modelled by means of discrete duration scenarios. The model was shown to continue to produce high-quality results for the case where disruption realisations are not sampled exactly from these scenarios. Therefore, we can say that the model is relatively *robust* to deviations in its input (at least with respect to the disruption data). In the same vein, the model has also been shown to be robust against deviations from the one-disruption assumption.

4. The branch-and-bound procedure that we have developed is several orders of magnitude faster than a general IP-solver. Nevertheless, the size of the scheduling instances that can be solved to guaranteed optimality remains limited and, especially for large float values, seems very little amenable to algorithmic speedup. This is not illogical in view of the limited size of problems solvable by other combinatorial optimisation approaches to scheduling under uncertainty (see Daniels and Carrillo 1997, Daniels and Kouvelis 1995, and Kouvelis and Yu 1997) and the additional complication that optimal schedules need not (and will generally not) be active. Further research is in order if realistically sized scheduling problems are to be dealt with. We are convinced that the insights provided in this paper can serve as guidelines in this process.

Acknowledgments

This research has been partially supported by project OT/03/14 of the Research Fund K.U.Leuven and project G.0109.04 of the Research Programme of the Fund for Scientific Research – Flanders (Belgium) (F.W.O.-Vlaanderen).

Appendices

Appendix A Counterexample for convexity of lb_1

We examine the behaviour of lb_1 for a scheduling problem with $N\backslash J_h$ containing two jobs, as a function of f for $f = 0$ to $f = 3$, if $c_1 = c_2 = 1$, $p_1 = 0.99$, $p_2 = 0.01$, $\Psi_1 = \{1, 2, 3\}$, $\Psi_2 = \{50, 51, 52\}$, and all g_{ik} equal. For successive values of $f = 0, 1, 2, 3$, we have $lb_1 = 0.51, 0.5, 0.33, 0$, such that the speed of descent increases with f, and the function cannot be convex.

Appendix B Description of the algorithmic implementation

The branch-and-bound algorithm described in this paper has been implemented along the lines of the following pseudo-code.

```
procedure BB()
     level=0;
     nrnodes=0;
NEW_LEVEL:
     level++;
     nr_alternatives_explored[level]=0;
     generate_and_order_alternatives();
NEW_ALTERNATIVE:
     nr_alternatives_explored[level]++;
     nrnodes++;
     implement_next_branching_alternative();
     if (fathomed) goto BACKTRACK;
     if (level<n) goto NEW_LEVEL;
FEASIBLE_SOLUTION:
    evaluate_objective_and_update_incumbent();
BACKTRACK:
    undo_branching_decisions_at_current_level();
    level--;
    if (nrexplored[level+1]<n-level)
            level++;
            goto NEW_ALTERNATIVE;
    else if (level>0) goto BACKTRACK;
FINISH:
    return;
```

References

Adiri, I., J. Bruno, E. Frostig, A.H.G. Rinnooy Kan. 1989. Single machine flow-time scheduling with a single breakdown. *Acta Informatica* **36** 679–696.

Ahuja, R., T. Magnanti, J. Orlin. 1993. *Network flows.* Prentice-Hall.

Akturk, M., E. Gorgulu. 1999. Match-up scheduling under a machine breakdown. *Eur. J. Oper. Res.* **112** 81–97.

Aytug, H., M. Lawley, K. McKay, S. Mohan, R. Uzsoy. 2004. Executing production schedules in the face of uncertainties: a review and some future directions. *Eur. J. Oper. Res.* in press.

Bean, J., J. Birge, J. Mittenthal, C. Noon. 1991. Match-up scheduling with multiple resources, release dates and disruptions. *Oper. Res.* **39** 470–483.

Bruno, J., E. Coffmann, R. Sethi. 1974. Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* **17** 382–387.

Calhoun, K., R. Deckro, J. Moore, J. Chrissis, J.V. Hove. 2002. Planning and re-planning in project and production planning. *Omega* **30** 155–170.

Daniels, R., J. Carrillo. 1997. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Trans.* **29** 977–985.

Daniels, R., P. Kouvelis. 1995. Robust scheduling to hedge against processing time uncertainty in single-stage production. *Mgmt. Science* **41** 363–376.

Dasdan, A., R. Gupta. 1998. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* **17** 889–899.

Hagstrom, J. 1988. Computational complexity of PERT problems. *Networks* **18** 139–147.

Hall, N., C. Potts. 2004. Rescheduling for new orders. *Oper. Res.* **52** in press.

Herroelen, W., R. Leus. 2004. The construction of stable project baseline schedules. *Eur. J. Oper. Res.* **156** 550–565.

Kanet, J., V. Sridharan. 2000. Scheduling with inserted idle time: problem taxonomy and literature review. *Oper. Res.* **48** 99–110.

Karp, R. 1978. A characterization of the minimum cycle mean in a digraph. *Discrete Math.* **23** 309–311.

Kouvelis, P., G. Yu. 1997. *Robust discrete optimization and its applications.* Kluwer Academic Publishers.

Leon, V., S. Wu, R. Storer. 1994. Robustness measures and robust scheduling for job shops. *IIE Trans.* **26** 343–362.

Leus, R. 2003. The generation of stable project plans. Complexity and exact algorithms. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium.

Leus, R., W. Herroelen. 2004. The complexity of machine scheduling for stability with a single disrupted job. *Oper. Res. Lett.* in press.

Lutz, C., K. Davis, M. Sun. 1998. Determining buffer allocation and size in production lines using tabu search. *Eur. J. Oper. Res.* **106** 301–316.

Mehta, S., R. Uzsoy. 1998. Predictable scheduling of a job shop subject to breakdowns. *IEEE Trans. on Robotics and Automation* **14** 365–378.

O'Donovan, R., R. Uzsoy, K. McKay. 1999. Predictable scheduling a single machine with breakdowns and sensitive jobs. *Internat. J. Prod. Res.* **37** 4217–4233.

Papadopoulos, H., M. Vidalis. 2001. A heuristic algorithm for the buffer allocation in unreliable unbalanced production lines. *Comput. Indust. Eng.* **41** 261–277.

Parker, R., R. Rardin. 1988. *Discrete optimization.* Academic Press.

Pinedo, M. 2002. *Scheduling. Theory, algorithms, and systems.* Prentice-Hall.

Raheja, A., V. Subramaniam. 2002. Reactive recovery of job shop schedules – a review.

Internat. J. of Advanced Manuf. Tech. **19** 756–763.

Rangsaritratsamee, R., W. Ferrel, M. Kurz. 2004. Dynamic rescheduling that simultaneously considers efficiency and stability. *Comput. Indust. Eng.* **46** 1–15.

Sevaux, M., K. Sörensen. 2003. A genetic algorithm for robust schedules in a just-in-time environment. Res. rep. LAMIH/SP-2003-1, Univ. Valenciennes, France.

Stork, F. 2001. Stochastic resource-constrained project scheduling. PhD thesis, TU Berlin, Berlin, Germany.

Wu, S., H. Storer, P.-C. Chang. 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Comput. Oper. Res.* **20** 1–14.

Yáñez, J., J. Ramírez. 2003. The robust coloring problem. *Eur. J. Oper. Res.* **148** 546–558.