KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

Sequential testing of *n*-out-of-*n* systems: precedence theorems and exact methods

Rostami S, Creemers S, Wei W, Leus R.

KBI_1806

Sequential testing of n-out-of-n systems: Precedence theorems and exact methods

Salim Rostami[1,2], Stefan Creemers[1], Wenchao Wei[3] and Roel Leus[2*]

[1]IÉSEG School of Management (LEM-CNRS 9221), Lille, France
[2]ORSTAT, KU Leuven, Leuven, Belgium
[3]School of Economics and Management, Beijing Jiaotong University, Beijing, China

Abstract: The goal of sequential testing is to discover the state of a system by testing its components one by one. We consider n-out-of-n systems, which function only if all n components work. The testing continues until the system state (up or down) is identified. The tests have known execution costs and failure probabilities and may be subject to precedence constraints. The objective is to find a sequence of tests that minimizes the total expected cost of the diagnosis. We show how to strengthen the precedence graph without losing all optimal solutions. We examine different formulations for the problem, and propose a dynamic-programming (DP) and a branch-and-price algorithm. Our computational results show that our DP noticeably outperforms the state of the art. Using a novel memory management technique, it significantly increases the size of the instances that can be solved to optimality under practical memory and time limits.

Keywords: sequencing, system testing, precedence theorems, dynamic programming, branch-and-price.
History: Submitted April 2018.

1 Introduction and related work

A multi-component machine, e.g., a missile radar set, that is inactive except during a state of military alert, undergoes periodic checkups. The state of the machine (working or failing) depends on the state of its components. In what order should the components be tested in order to detect the state of the machine with minimum efforts with respect to cost and/or time? With these challenges in mind for preventive and corrective maintenance, the importance of *sequential testing* was highlighted by the U.S. Air Force more than sixty years ago [22]. The goal of sequential testing is to discover the state of a system by testing its components one by one. The testing continues until the system state is identified. Each test has a cost and a success probability. The objective is to find a sequence of tests that minimizes the total expected cost of the diagnosis.

As equipment becomes more complex and expensive, the number of required tests and the inspection costs increase, and consequently, the sequential diagnosis becomes more costly.

*Corresponding author. E-mail: roel.leus@kuleuven.be, tel.: +32 16 32 69 67.

From another perspective, as automated robotic machines replace human mechanics and quality control operators, it becomes more vital to be able to prescribe a desirable sequence of inspections for machines [22]. In many practical applications, including rapid diagnosis in time-critical situations, e.g., toxic chemical identification [2], R&D project scheduling [11, 14], quality control of manufactured products [17], container inspection operations at borders [25], and so on, the underlying problem incorporates sequential testing. Different tests may require different resources (e.g., manpower, electricity, etc.) or may entail other costs (e.g., pain to a patient). Therefore, examiners may try to postpone the more costly tests. Moreover, the tests may be subject to precedence constraints. In pharmaceutical drug testing, for instance, animal (preclinical) tests precede human (clinical) tests [1].

An *n-out-of-n* (also n:n or serial) system is up if all its n components are functioning. Hence, its testing terminates when the first failure is revealed. In a 1:n (parallel) system, on the other hand, a single working component guarantees the functionality of the whole system. From an optimization perspective, serial and parallel systems are equivalent: an optimal solution to a 1:n system with success probabilities p_i is also optimal to an n:n system with success probabilites $1 - p_i$ [36]. A k:n system is up if at least k tests are successful, and is down if $(n - k + 1)$ tests fail. Sequential testing of k:n systems is studied by Wei et al. [39]. The state of a *modular* system, where each module represents a multi-part component, is up if all its modules are functional. A module is working if at least one of its parts functions. The modules and the tests within each module may be precedence-related. Coolen et al. [9] and Creemers et al. [11] consider modular systems in project scheduling.

Various exact approaches have been proposed for special cases of sequential testing. It is shown that serial and parallel systems without precedence constraints are polynomially solvable [4, 6, 21, 22, 27]. A polynomial algorithm also exists for arbitrary k without precedence constraints [3, 5, 7, 32, 33]. Serial systems are polynomially solvable when the precedence graph is a forest [20] or series-parallel [28]. An optimal diagnosis procedure is provided in [8] for k:n systems with parallel-chain precedence constraints. Wagner and Davis [38] propose an integer-linear formulation for a single-item discrete sequential search problem, which is in fact a serial system known to be down. For an extensive literature review on sequential testing we refer to Ünlüyurt [36]. More recently, Daldal et al. [12] have offered approximation algorithms for sequential batch-testing of serial systems.

The subject of this article is sequential testing of n:n systems with a general precedence graph, which is known to be strongly NP-hard [24]. To the best of our knowledge, no research has been specifically devoted to developing exact algorithms for this problem. The main contributions of this work are the following. (1) We show how to strengthen the

precedence graph in a pre-processing step. (2) We develop a linear formulation for the problem obtained via Dantzig-Wolfe decomposition [13] of a compact nonlinear formulation. (3) A dynamic-programming (DP) and a branch-and-price (B&P) algorithm are devised as solution methods. The computational results indicate that the DP procedure is faster than B&P. Compared to the existing algorithms, our method significantly increases (by up to 100%) the size of the instances that can be solved to optimality within practical memory and time limits. (4) In the process, we show that a relaxation of the pricing problem of our B&P can be solved in polynomial time. We use this result to solve the pricing problem more efficiently.

Below, we first formally define the problem in Section 2. Section 3 is devoted to three precedence theorems for strengthening the precedence graph. We propose a DP and a B&P algorithm in Sections 4 and 5, respectively. Computational results are provided in Section 6. We conclude in Section 7.

2 Problem statement

We consider a set N of n tests (or inspections) with pre-specified testing costs $c_i \geq 0$, $i \in N$, and success probabilities $p_i \in [0, 1]$. The set E is a partial order on N, representing the precedence constraints. A test can be performed only if all its predecessors in E are done. In a sequential diagnosis, tests are performed one at a time until the correct state of the system is identified. The system is functioning when all the n tests are successful, and it is known to be down when the first failure is revealed.

A solution to the sequential testing problem is a decision policy that decides which test should be started next given a history of previously performed tests and their results. For serial and parallel systems, there is a full order (sequence) of tests that is optimal [39]. This does not necessarily hold for k:n systems. A full order is feasible iff it extends E. The objective is to find a sequence (list) $L = (l_1, l_2, ..., l_n)$ of the tests that minimizes the total expected cost of the diagnosis $\sum_{j=1}^{n}(\prod_{i=1}^{j-1} p_{l_i})c_{l_j}$, where $\prod_i p_i$ over an empty set equals 1.

Example: Consider the instance depicted in Figure 1 with four tests and precedence graph $G(N, E)$. The sequence $L^1 = (1, 2, 3, 4)$ is a feasible solution with objective value $1 \times 5 + 0.9 \times 8 + 0.45 \times 40 + 0.405 \times 1 = 30.605$, while $L^* = (1, 4, 2, 3)$ is an optimal sequence with objective value 8.42. □

Following Potts [30], we describe a compact formulation (CF) based on "linear ordering"

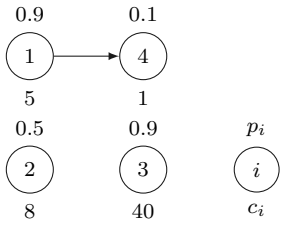


Figure 1: A problem instance

variables x_{ij}, $i, j \in N$, where $x_{ij} = 1$ if test i precedes test j, and $x_{ij} = 0$ otherwise.

$$\text{(CF)} \qquad \min \quad \sum_{j=1}^{n} c_j \prod_{i=1}^{n} [(p_i - 1)x_{ij} + 1] \tag{2.1}$$

$$\text{s.t.} \quad x_{ij} + x_{ji} = 1 \qquad \forall \{i, j\} \subset N \tag{2.2}$$

$$x_{ji} + x_{ik} - x_{jk} \leq 1 \qquad \forall \{i, j, k\} \subset N \tag{2.3}$$

$$x_{ij} = 1 \qquad \forall (i, j) \in E \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{2.5}$$

Constraint (2.2) imposes anti-symmetry. Wagner and Davis [38] show that Constraint (2.3) is sufficient to enforce the transitivity property and to eliminate cycles of any size. Constraint (2.4) enforces the precedence constraints in E.

Although correct, formulation CF is not practical due to its nonlinear objective. Following, we employ Dantzig-Wolfe decomposition [13] to linearize CF. For more information on applying Dantzig-Wolfe decomposition to mixed integer programs we refer to [37]. For any $i \in N$, let $\Gamma_i = \{x_i^m \in \{0,1\}^n \mid x_{ji}^m = 1, \forall (j,i) \in E\}$ be the set of ρ_i points $x_i^m = \{x_{1i}^m, x_{2i}^m, \ldots, x_{ni}^m\}$ that satisfy (2.4) and (2.5). Every x_i^m corresponds to a set $C_i^m \subset N$ of candidate predecessors of test i, where $j \in C_i^m$ iff $x_{ji}^m = 1$. Intuitively, C_i^m is feasible iff for any $(j,i) \in E : j \in C_i^m$, for any $(i,k) \in E : k \notin C_i^m$, and $i \notin C_i^m$. We define $P_i^m = \prod_{j \in C_i^m} p_j$ as the joint success probability of C_i^m.

Example (continued): Considering the previous example, for test 1, there are four different subsets, namely $C_1^1 = \emptyset$, $C_1^2 = \{2\}$, $C_1^3 = \{3\}$, $C_1^4 = \{2,3\}$. The corresponding joint probabilities are $P_1^1 = 1$, $P_1^2 = 0.5$, $P_1^3 = 0.9$, $P_1^4 = 0.45$. □

Constraints (2.4) and (2.5) can be dropped from CF if we restrict the solution space to the

sets Γ_i, i.e., if we impose

$$\forall i \in N : x_i = \sum_{m=1}^{\rho_i} z_i^m x_i^m, \text{ with } \sum_{m=1}^{\rho_i} z_i^m = 1 \text{ and } z_i^m \in \{0,1\},$$

where the decision variable z_i^m is 1 if the tests in C_i^m precede i, and 0 otherwise. Using this transformation and by replacing $\prod_{i=1}^{n}[(p_i-1)x_{ji}^m+1]$ by P_i^m in the thus-obtained objective function, we reach the following integer linear formulation (ILF).

$$\text{(ILF)} \quad \min \quad \sum_{i=1}^{n} c_i \sum_{m=1}^{\rho_i} P_i^m z_i^m \tag{2.6}$$

$$\text{s.t.} \quad \sum_{m=1}^{\rho_j} z_j^m x_{ij}^m + \sum_{m=1}^{\rho_i} z_i^m x_{ji}^m = 1 \qquad \forall \{i,j\} \subset N \tag{2.7}$$

$$\sum_{m=1}^{\rho_j} z_i^m x_{ji}^m + \sum_{m=1}^{\rho_k} z_k^m x_{ik}^m - \sum_{m=1}^{\rho_k} z_k^m x_{jk}^m \leq 1 \qquad \forall \{i,j,k\} \in N \tag{2.8}$$

$$\sum_{m=1}^{\rho_i} z_i^m = 1 \qquad \forall i \in N \tag{2.9}$$

$$z_i^m \in \{0,1\} \qquad \forall i \in N,\ m \in \{1,\ldots,\rho_i\} \tag{2.10}$$

The number of variables z_i^m is exponential in n. To overcome this difficulty, in Section 5, we propose an exact algorithm based on column generation for solving ILF.

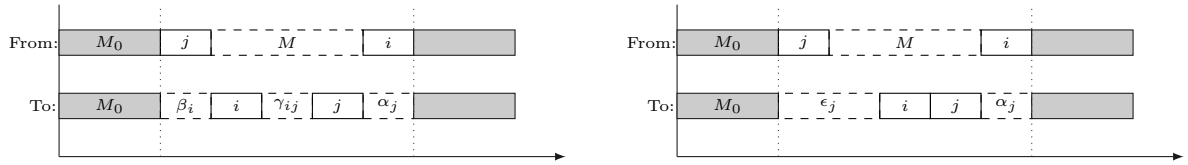
3 Precedence theorems

A precedence constraint $(i,j) \in N \times N$ is said to be *dominant* iff there is at least one optimal solution with i before j. Given an instance $G(N,E)$, let D be the set of all dominant constraints. Precedence theorems may help us identify some of the constraints in D. Note that since every feasible solution satisfies the constraints in E, we have $E \subseteq D$.

We know that when $E = \emptyset$, a sequence $L = (l_1, l_2, \ldots, l_n)$ is optimal [36] if

$$\frac{c_{l_1}}{1-p_{l_1}} \leq \frac{c_{l_2}}{1-p_{l_2}} \leq \cdots \leq \frac{c_{l_n}}{1-p_{l_n}}. \tag{3.11}$$

The condition is intuitive: for a series system, it is advantageous to perform tests with low cost and high failure probability first. If $c_i/(1-p_i) \leq c_j/(1-p_j)$ then if j precedes i in a given sequence, we can swap the two tests and the thus-obtained sequence cannot have a higher objective value than the initial sequence. We say the second sequence *dominates* the original.



(a) Swap (b) Insert-after

Figure 2: Swap and insert-after strategies when $E \neq \emptyset$

We now develop an extension of this rule for general E. Let $B_i = \{j \in N \mid (j,i) \in E\}$ and $A_i = \{j \in N \mid (i,j) \in E\}$ be the sets of tests that according to E should be done before and after test i, respectively. For any $\{i,j\} \subset N$ with $(i,j),(j,i) \notin E$, we prove the following three theorems. Remark that for such a pair, $G(N, E \cup \{(i,j)\})$ is acyclic.

Theorem 1. *If for any $k \in \{i\} \cup B_i \setminus B_j$ and $q \in \{j\} \cup A_j \setminus A_i$ we have $c_k/(1-p_k) \leq c_q/(1-p_q)$, then $(i,j) \in D$.*

Proof: We need to show that not only (i,j) but also the associated transitive constraints are dominant. Consider the set $\{(k,q) \mid k \in B_i, q \in A_j\}$ of the pairs that are related to (i,j). For general E and for any $k \in B_i \cap B_j$ and $q \in A_i \cap A_j$, the pair $(k,q) \in E$ and consequently is respected by every feasible solution. Therefore, we only need to check the condition $c_k/(1-p_k) \leq c_q/(1-p_q)$ for any $k \in \{i\} \cup B_i \setminus B_j$ and $q \in \{j\} \cup A_j \setminus A_i$. If the condition holds for every such pair, then the pair (i,j) and its corresponding transitive pairs are dominant. □

Below, we use the so-called "swap" and "insert-after" strategies [23, 31] to obtain two more precedence theorems. Given a feasible sequence with j before i and $(j,i) \notin E$, swapping the two jobs or inserting j after i (together with the corresponding transitive changes), might lead to an improvement in the objective. The two strategies aim to find closed-form conditions to identify circumstances where such improvements are guaranteed. An illustration of these strategies for general E is provided in Figure 2, where $\beta_i = (M \cap B_i)$, $\alpha_j = (M \cap A_j)$, $\gamma_{ij} = M \setminus (\alpha_j \cup \beta_i)$ and $\epsilon_j = M \setminus \alpha_j$, with M the set of intermediate tests between i and j. "From" represents any feasible sequence, and "To" is the resulting sequence after swapping j and i or inserting j after i. Following the Swap strategy, a sufficient condition for the dominance of (i,j) has the structure

$$\mathrm{LB}(\mathrm{OI}(j)) \geq \mathrm{UB}(\mathrm{OD}(i)) + \mathrm{UB}(\mathrm{OD}(\gamma_{ij})) + \mathrm{UB}(\mathrm{OD}(\beta_i)), \tag{3.12}$$

where $\mathrm{LB}(\cdot)$ and $\mathrm{UB}(\cdot)$ are lower and upper bound functions. For any $X \subseteq N$, we define $\mathrm{OI}(X) = \sum_{l \in X} \max\{0, c_l \prod_{k \prec_{\mathrm{To}} l} p_k - c_l \prod_{k \prec_{\mathrm{From}} l} p_k\}$ as the objective improvement of the

tests in X after the changes, and $\mathrm{OD}(X) = \sum_{l \in X} \max\{0, c_l \prod_{k \prec_{\mathrm{From}} l} p_k - c_l \prod_{k \prec_{\mathrm{To}} l} p_k\}$ as their objective degradation, where $i \prec_L j$ indicates that i precedes j in L. For simplicity of notation, we use $\mathrm{OI}(k)$ instead of $\mathrm{OI}(\{k\})$ for all singleton sets $X = \{k\}$, with $k \in N$. Note that $\mathrm{UB}(\mathrm{OD}(\alpha_i))$ and $\mathrm{LB}(\mathrm{OI}(\alpha_i))$ are set to 0. A variant of Condition (3.12) can be used for the insert-after strategy by replacing $\mathrm{UB}(\mathrm{OD}(\gamma_{ij})) + \mathrm{UB}(\mathrm{OD}(\beta_i))$ by $\mathrm{UB}(\mathrm{OD}(\epsilon_j))$.

If a pair (i, j) of tests satisfies Condition (3.12) for *every* feasible M, and if $G(N, E \cup \{(i, j)\})$ is acyclic, then if j precedes i in a given schedule, we can exchange the two tests without increasing the cost. Thus, for a set of test pairs $\{(i, j), (k, q), \dots\}$ that each satisfy one or both of the swap and insert-after dominance conditions, any optimal schedule that is not compatible with one or more of these pairs cannot be deteriorated by making as many interchanges as necessary to obtain an optimal schedule that respects all the pairs. Let $S \subseteq D$ with $S \supseteq E$ be the set of dominant precedence constraints that are previously identified. Starting from $S = E$, we iteratively identify dominant precedence constraints, which are then added to S. The iterative extension of S stops only when we can no longer identify new dominant constraints. We conclude:

Proposition 1. *There is at least one optimal solution that respects all the precedence constraints in S iff the graph $G(N, S)$ is acyclic.*

Similarly to B_i and A_i, we define B_i^S and A_i^S based on S instead of E. Moreover, for any $X \subseteq N$, we define $P(X) = \prod_{i \in X} p_i$ and $\mathrm{CF}[X]$ is the subproblem where N and E in CF are replaced by X and $\{(i, j) \in E \mid i, j \in X\}$, respectively. Finally, $\lambda(B_i^S)$, $\theta(\gamma_{ij})$, and $\phi(\epsilon_j)$ are upper bounds to the subproblems $\mathrm{CF}[B_i^S]$, $\max_{\gamma_{ij} \subseteq N \setminus (\{j\} \cup B_j^S \cup A_j^S \cup \{i\} \cup B_i^S \cup A_i^S)} \{\mathrm{CF}[\gamma_{ij}]\}$, and $\max_{\epsilon_j \supseteq B_i^S \setminus B_j^S, \epsilon_j \subseteq N \setminus (\{j\} \cup B_j^S \cup A_j^S \cup \{i\} \cup A_i^S)} \{\mathrm{CF}[\epsilon_j]\}$, respectively, where γ_{ij} and ϵ_j are precedence-feasible subsets of tests.

Theorem 2. *If*

$$\begin{aligned} & c_j(1 - p_i)P(N \setminus (\{j\} \cup A_j^S \cup \{i\} \cup A_i^S)) \geq \\ & \quad (c_i P(B_j^S \cup B_i^S) + P(B_j^S)\lambda(B_i^S))(1 - p_j P[N \setminus (\{j\} \cup B_j^S \cup \{i\} \cup B_i^S \cup A_i^S)]) \\ & \quad + P(B_j^S \cup B_i^S)\theta(\gamma_{ij}) \max\{0, p_i - p_j P(A_j^S)\}, \end{aligned} \tag{3.13}$$

then $(i, j) \in D$.

Proof: For Condition (3.12) for the swap strategy we have

$$\mathrm{OI}(j) = \max\{0, P(M_0)c_j - P(M_0)P(\beta_i)p_i P(\gamma_{ij})c_j\}$$

$$
\begin{aligned}
&= P(M_0)c_j(1-p_iP(\beta_i)P(\gamma_{ij})) \\
&\geq c_j(1-p_i)P[N \setminus (\{j\} \cup A_j^S \cup \{i\} \cup A_i^S)],
\end{aligned}
$$

where M_0 is the set of tests that precede j in From sequence in Figure 2a.

$$
\begin{aligned}
\mathrm{OD}(i) &= \max\{0, P(M_0)P(\beta_i)c_i - P(M_0)p_jP(M)c_i\} \\
&= c_iP(M_0)P(\beta_i)(1-p_jP(M \setminus \beta_i)) \\
&\leq c_iP(B_j^S \cup B_i^S)(1-p_jP[N \setminus (\{j\} \cup B_j^S \cup \{i\} \cup B_i^S \cup A_i^S)]). \\
\mathrm{OD}(\beta_i) &\leq \max\{0, P(M_0)\lambda(\beta_i) - P(M_0)p_jP(M \setminus \beta_i)\lambda(\beta_i)\} \\
&= P(M_0)\lambda(\beta_i)(1-p_jP(M \setminus \beta_i)) \\
&\leq P(B_j^S)\lambda(B_i^S)(1-p_jP[N \setminus (\{j\} \cup B_j^S \cup \{i\} \cup B_i^S \cup A_i^S)]). \\
\mathrm{OD}(\gamma_{ij}) &\leq \max\{0, P(M_0)P(\beta_i)p_i\theta(\gamma_{ij}) - P(M_0)p_jP(\alpha_j)P(\beta_i)\theta(\gamma_{ij})\} \\
&= P(M_0)P(\beta_i)\theta(\gamma_{ij})\max\{0, p_i - p_jP(\alpha_j)\} \\
&\leq P(B_j^S \cup B_i^S)\theta(\gamma_{ij})\max\{0, p_i - p_jP(A_j^S)\}.
\end{aligned}
$$

By substituting the expressions above in (3.12), we have (3.13). □

Theorem 3. *If*

$$
\begin{aligned}
c_j(1-p_i)P(N \setminus (\{j\} \cup A_j^S \cup \{i\} \cup A_i^S)) \geq \\
(c_iP(B_j^S \cup B_i^S) + P(B_j^S)\phi(\epsilon_j))(1-p_jP(A_j^S)),
\end{aligned} \tag{3.14}
$$

then $(i,j) \in D$.

Proof: For Condition (3.12) for the insert-after strategy we have

$$
\begin{aligned}
\mathrm{OI}(j) &\geq c_j(1-p_i)P[N \setminus (\{j\} \cup A_j^S \cup \{i\} \cup A_i^S)]. \\
\mathrm{OD}(i) &= \max\{0, P(M_0)P(\epsilon_j)c_i - P(M_0)p_jP(M)c_i\} \\
&= c_iP(M_0)P(\epsilon_j)(1-p_jP(\alpha_j)) \\
&\leq c_iP(B_j^S \cup B_i^S)(1-p_jP(A_j^S)). \\
\mathrm{OD}(\epsilon_j) &\leq \max\{0, P(M_0)\phi(M \setminus \alpha_j) - P(M_0)p_jP(\alpha_j)\phi(\epsilon_j)\} \\
&= P(M_0)\phi(\epsilon_j)(1-p_jP(\alpha_j)) \\
&\leq P(B_j^S)\phi(\epsilon_j)(1-p_jP(A_j^S)).
\end{aligned}
$$

By substituting the expressions above in (3.12) we have (3.14). □

There is an obvious trade-off between the quality of the bounds $\lambda(B_i^S)$, $\theta(\gamma_{ij})$ and $\phi(\epsilon_j)$, and the time complexity of their computation. Here, we opt for obtaining these bounds in polynomial time. We obtain $\lambda(B_i^S)$ by ordering the tests in non-decreasing order of $c_i/(1-p_i)$ while respecting the precedence constraints. That is, starting from the first position of an empty list, we fill the n positions, iteratively. At each of the n decision points, we choose the eligible test with lowest $c_i/(1-p_i)$. For $\theta(\gamma_{ij})$ and $\phi(\epsilon_j)$, one straightforward approach is to order the tests in non-increasing order of $c_i/(1-p_i)$, ignoring the precedence constraints.

Example (continued): For the previous example, we obtain $(2,1,4,3)$ for $\lambda(N)$ with the objective value 12.75. As an upper bound to $\max_{X\subseteq N}\{\mathrm{CF}[X]\}$ (as for θ and ϕ bounds), on the other hand, we obtain $(3,1,2,4)$ with the objective value 51.385. For this example, we can show that Theorems 1 and 3 identify $(1,3)$, $(2,3)$ and $(4,3)$ as dominant, while Theorem 2 identifies $(4,2)$ and $(4,3)$. The identified constraints are respected by the optimal solution $L^* = (1,4,2,3)$. □

In the remainder, we assume that the input precedence networks have been tightened using the precedence theorems, but we maintain the notation E instead of S for the set of precedence constraints. We also assume that the test indices form a topological order of $G(N,E)$.

4 DP algorithm

In this section, we propose a DP algorithm for solving the sequential testing problem for n:n systems. In the remainder of the text, we will refer to this algorithm as DPn. The recursion of DPn (Section 4.1) is close to the recursion in [9, 39], but tuned specifically towards the n:n problem. We use an improved version of the memory management technique in [31] (Section 4.2). This technique enables us to solve significantly larger instances (by up to 100%) than before within practical time and memory limits.

4.1 DP recursion

Each state is defined as a set $Y \subseteq N$ of to-be-scheduled tests. Every state Y represents a sequential testing subproblem, where the first $n-|Y|$ positions of the sequence are filled, and we decide the test that occupies the $(n-|Y|+1)^{\text{th}}$ position. The state space Φ contains

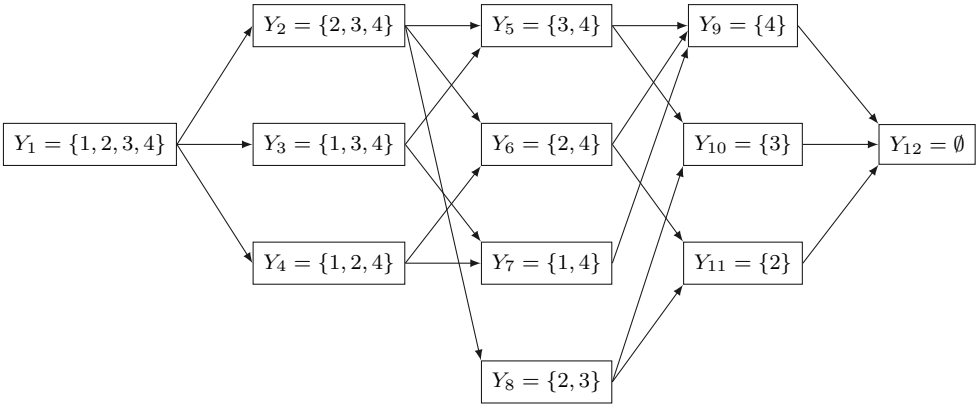


Figure 3: The state space of the example in Figure 1

all the feasible states. A state Y is feasible if it respects the precedence constraints, i.e., $\forall i \in Y : A_i \subset Y$. Given a state $Y \in \Phi$, let $\xi(Y) = \{i \in Y \mid B_i \cap Y = \emptyset\}$ be the set of tests that are eligible to be tested. Hence, $\xi(Y)$ contains all the possible decisions to be made in Y. Selecting test $i \in \xi(Y)$ leads to $Y \setminus \{i\}$ as the immediate succeeding state. The value function of any state Y can be computed via the backward recursion

$$G(Y) = \min_{i \in \xi(Y)} \{c_i + p_i G(Y \setminus \{i\})\}. \tag{4.15}$$

Starting from the unique final state $Y = \emptyset$ with $G(\emptyset) = 0$, DPn iteratively calculates the objective value corresponding to the preceding states.

Example (continued): Figure 3 depicts the state space of the previous example. The objective value of Y_9 is $c_4 + p_4 \times G(\emptyset) = 1$, while $G(Y_2) = \min\{c_2 + p_2 \times G(Y_5), c_3 + p_3 \times G(Y_6), c_4 + p_4 \times G(Y_8)\} = 3.8$. Finally, $G(Y_1) = 8.42$ is the optimal objective value; the unique optimal solution is $L^* = (1, 4, 2, 3)$. □

4.2 DP memory management

In this section we elaborate an improved version of the memory management technique that is proposed in [31] for precedence-constrained single machine scheduling with weighted tardiness objective. The method partitions the state space into sets of states with equal cardinality, i.e., it divides Φ into sets $\Upsilon^f = \{Y \in \Phi : |Y| = f\}$, with $0 \leq f \leq n$. In Figure 3, each Υ^f corresponds to one "column" of states, e.g., $\Upsilon^2 = \{Y_5, Y_6, Y_7, Y_8\}$. For a given $Y \in \Phi$, let $Q(Y) = \{i \in N \setminus Y \mid A_i \subseteq Y\}$ be the set of tests in $N \setminus Y$ with all successors in Y. Hence, for any $i \in Q(Y)$, $Y \cup \{i\}$ is a feasible state. For a given f $(0 < f \leq n)$ we

Algorithm 1 Υ^f generation subroutine

Input: Υ^{f-1} in decreasing binary order **and** $\forall Y \in \Upsilon^{f-1} : Q(Y)$ in decreasing index order
Output: Υ^f in decreasing binary order
$a = u = 0$
for $i = 1$ **to** $|\Upsilon^{f-1}|$ **do**
  if i==1 **then**
    $q = 1$
  else
    $q = \min\{j \in \mathbb{N} \mid Q(\Upsilon_i^{f-1})[j] < \max\{i \in \Upsilon_a^{f-1} \setminus \Upsilon_i^{f-1}\}\}$
  end if
  for $j = q$ **to** $|Q(\Upsilon_i^{f-1})|$ **do**
    $u = u + 1$
    $\Upsilon_u^f = \Upsilon_i^{f-1} \cup Q(\Upsilon_i^{f-1})[j]$
    $a = i$
  end for
end for
return Υ^f

have

$$\Upsilon^f = \bigcup_{Y \in \Upsilon^{f-1}} \{Y \cup \{i\} \mid i \in Q(Y)\} \tag{4.16}$$

and $\Upsilon^0 = \{\emptyset\}$. Therefore, the generation of Υ^f depends only on Υ^{f-1}. Moreover, according to (4.15), for any state $Y \in \Upsilon^f$ the objective value $G(Y)$ is determined by the objective value of states in Υ^{f-1}. We conclude that, once Υ^f is created and the objective value of its members is computed, the set Υ^{f-1} can be discarded from memory. This means that at any time during the execution of DP, we need to store at most two sets of states (Υ^{f-1} and Υ^f).

One drawback of the method of Rostami et al. [31] is that for each newly generated state, it has to be checked if the state has been generated before. If a state already exists in the set of generated states, then we only need to update its objective value. In this article we propose an improved version of the technique that does not require the aforementioned checks and is consequently faster.

Algorithm 1 elaborates a subroutine that receives Υ^{f-1} and generates Υ^f, where $Q(Y)[i]$ is the i^{th} element in $Q(Y)$, Υ_i^f is the i^{th} element of Υ^f and u is a counter of the states in Υ^f that are previously generated. With each state Y, we associate $\tau(Y) = \sum_{i \in Y} 2^{i-1}$ as its *binary representation*. The elements of the resulting Υ^f are generated in decreasing order of their binary representation if the states in Υ^{f-1} are processed in decreasing binary order, and for each state $Y \in \Upsilon^{f-1}$, the tests in $Q(Y)$ are scanned in decreasing index order.

Example (continued): In Figure 3 and for Υ^2, the states are in decreasing binary order: $\Upsilon_1^2 = Y_5$, $\Upsilon_2^2 = Y_6$, $\Upsilon_3^2 = Y_7$, and $\Upsilon_4^2 = Y_8$ with $\tau(Y_5) = 12$, $\tau(Y_6) = 10$, $\tau(Y_7) = 9$ and $\tau(Y_8) = 6$. The states that are generated from Y_5, are in decreasing binary order if we scan

the tests in $Q(Y_5) = \{1, 2\}$ in decreasing index order, meaning $Q(Y_5)[1] = 2$ and $Q(Y_5)[2] = 1$. In this case, $Y_2 = Y_5 \cup \{2\}$ with $\tau(Y_2) = 14$ and $Y_3 = Y_5 \cup \{1\}$ with $\tau(Y_3) = 13$. □

Suppose the states Υ_{u-1}^f and Υ_u^f are generated from the states Υ_a^{f-1} and Υ_b^{f-1} $(a < b)$, respectively. We define $h = \max\{i \in \Upsilon_a^{f-1} \setminus \Upsilon_b^{f-1}\}$.

Theorem 4. *In Algorithm 1, the state $\Upsilon_b^{f-1} \cup \{i\}$ with $i \in Q(\Upsilon_b^{f-1})$ has already been generated iff $i \geq h$.*

Proof: We define $g = \min\{i \in Y_b^{f-1}\}$. It can be shown that $h > g$. Assume that Υ_u^f is produced by adding $Q(\Upsilon_b^{f-1})[q]$ to Υ_b^{f-1} and that $Q(\Upsilon_b^{f-1})[q] \geq h$. Since the test indices form a topological order of $G(N, E)$, we know that $B_g \cap \Upsilon_b^{f-1} = \emptyset$ and $g \notin A_{Q(\Upsilon_b^{f-1})[q]}$. Consequently, $Y^* = \Upsilon_b^{f-1} \setminus \{g\} \cup \{Q(\Upsilon_b^{f-1})[q]\}$ is a feasible state with cardinality $f - 1$. Since $\tau(Y^*) > \tau(\Upsilon_b^{f-1})$, there exists $c \leq a$ such that the state $\Upsilon_c^{f-1} = Y^*$. Knowing that $Y' = Y^* \cup \{g\}$, we conclude that Y' has already been generated. On the other hand, if $Q(\Upsilon_b^{f-1})[q] < h$, then $\Upsilon_u^f < \Upsilon_{u-1}^f$ and since the elements in Υ^f are generated in decreasing binary order, we conclude that Υ_u^f has not been generated before. □

Example (continued): For the states that are generated from Y_6, we have $h = \max\{i \in Y_5 \setminus Y_6\} = 3$ and $g = \min\{i \in Y_6\} = 2$. Since $Q(Y_6) = \{1, 3\}$, we conclude that the state $Y_6 \cup \{3\}$ is already generated (Y_2) from the state $Y_6 \setminus \{2\} \cup \{3\} = Y_5$. The state $Y_4 = Y_6 \cup \{1\}$, on the other hand, is new. □

Wei et al. [39] use *uniformly directed cuts* (*UDC*s) to structure the state space Φ. Each UDC $u \subseteq N$ is an inclusion-maximal set of incomparable tests, i.e., for any $\{i, j\} \in u$: $(i, j) \notin E \wedge (j, i) \notin E$. Let U be the set of all UDCs. With each UDC $u \in U$, Wei et al. associate a set $\sigma(u) \subset \Phi$ of states in which one or more tests in u are eligible to start and the tests in $N \setminus u$ are all ineligible or finished. It is shown in [10] that the sets $\sigma(u)$ are mutually exclusive and $\bigcup_{u \in U} \sigma(u) = \Phi$. Hence, U is a valid partition of Φ and allows, at each time during the optimization, to store and process only a subset of Φ, and to dismiss the states that will no longer be used. The number of UDCs is exponential in n, and they have to be enumerated prior to optimization. Our method, on the other hand, is not dependent on UDCs and combines the generation of the states with the calculation of their objective value during optimization. In Section 6, we empirically show that our memory management technique not only reduces the computational effort required to generate and search the state space, but also reduces the number of states that are stored in memory simultaneously. In other words, it is not only faster but also more memory-efficient.

5 Column generation

Column generation (CG) [26] is a technique for solving linear programs with a large number of variables. The algorithm consists of a *master* and a *pricing* problem. Here, the linear relaxation of ILF, denoted by LF, is the master problem. It is called *restricted* (RLF) when it includes only a subset of its variables (columns). RLF is solved iteratively, and at each iteration, the pricing problem determines whether a new variable exists that if added to the restricted master problem, will improve its objective value. If no such variable exists then an optimal solution to the current RLF is also optimal to LF. A heuristic algorithm (see Section 6.2) is used to obtain the initial columns of RLF.

5.1 The pricing problem

Let $\bar{\beta}_{ij}$, κ_{ijk} and α_i be the dual variables attached to Constraints (2.7), (2.8) and (2.9), respectively. The reduced cost of the variable z_i^m is then

$$P_i^m c_i - \alpha_i - \sum_{j \in N \setminus \{i\}} \bar{\beta}_{ij} x_{ji}^m - \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} \kappa_{ijk} x_{ji}^m.$$

The pricing problem then boils down to the question $\exists i, m : P_i^m c_i - \alpha_i - \sum_{j \in C_i^m} \beta_{ij} < 0$?, where $\beta_{ij} = \bar{\beta}_{ij} + \sum_{k \in N \setminus \{i,j\}} \kappa_{ijk}$. A new entering variable with negative reduced cost corresponds to a new set C_i^m. To find such variables, at each iteration, and for every $i \in N$, we solve the pricing problem

$$\text{(PR)} \qquad \min \quad F(N_i, \varsigma_i, \beta_i) = \varsigma_i \prod_{j \in N_i} p_j^{y_j} - \sum_{j \in N_i} \beta_{ij} y_j \tag{5.17}$$

$$\text{s.t.} \quad y_j \geq y_k \qquad \forall (j,k) \in E \tag{5.18}$$

$$y_j \in \{0,1\} \qquad \forall j \in N_i \tag{5.19}$$

where $N_i = N \setminus (B_i \cup A_i \cup \{i\})$ is the set of tests that are not precedence-related to i, and $\varsigma_i = c_i \prod_{j \in B_i} p_j$. The decision variable y_j is equal to 1 if j is selected and 0 otherwise. For given arguments, the objective value of an optimal solution $\mathbf{y}^*$ is denoted by $F(\mathbf{y}^*)$ or F^*. If there exists $i \in N$ such that $F(N_i, \varsigma_i, \beta_i; \mathbf{y}^*) < \alpha_i + \sum_{j \in B_i} \beta_{ij}$, then the subset $C_i^m = \{j \in N_i \mid y_j^* = 1\} \cup B_i$ and its corresponding variable z_i^m should enter RLF; otherwise, a current optimal solution to RLF is also optimal to LF.

5.1.1 A DP algorithm for solving PR

In this section, we develop a DP algorithm for solving the pricing problem PR. Henceforth, we refer to this algorithm as DPp. The problem bears similarities with the precedence-constrained knapsack problem. The main framework of DPp is also inspired by the DP algorithm of Samphaiboon and Yamada [34].

For the pricing problem PR$[N_i, \varsigma_i, \beta_i]$, a DP state is defined by (Y, Z), where $Y \subseteq N_i$ is a set of tests that may or may not be selected in the new set of predecessors of i, and $Z \subseteq (N_i \cup B_i) \backslash Y$ with $Z \supseteq B_i$ is the set of tests that are chosen to be in the set of predecessors of i. Each DP state (Y, Z) represents a subproblem PR$[Y, \varsigma, \beta_i]$, where $\varsigma = c_i \prod_{j \in Z} p_j$. Let Ω be the set of all feasible states. A state (Y, Z) is called feasible if it respects the precedence constraints in E, i.e., for each $j \in Y : B_j \subset (Y \cup Z)$ and for each $j \in Z : B_j \subset Z$. A state (Y, Z) is called *singleton* if $|Y| = 1$. We define $\delta(Y) = \min\{j \in Y\}$ as the test with minimum index in Y. Assuming that the test indices form a topological order of $G(N, E)$, we have $B_{\delta(Y)} \cap Y = \emptyset$. Considering a DP state (Y, Z), fixing $y_{\delta(Y)} = 1$ leads to $(Y \backslash \{\delta(Y)\}, Z \cup \{\delta(Y)\})$ and the choice $y_{\delta(Y)} = 0$ results in $(Y \setminus (\{\delta(Y)\} \cup A_{\delta(Y)}), Z)$. For a given state (Y, Z), the value function $\bar{F}$ computes the minimum reduced cost via the following recursion.

$$\begin{aligned} \bar{F}^*(Y, Z) = \min\{&\bar{F}^*(Y \setminus \{\delta(Y)\}, Z \cup \{\delta(Y)\}) + R(Z, Y), \\ &\bar{F}^*(Y \setminus (\{\delta(Y)\} \cup A_{\delta(Y)}), Z)\}, \end{aligned} \tag{5.20}$$

where $R(Y, Z) = c_i \prod_{j \in Z} p_j (p_{\delta(Y)} - 1) - \beta_{i,\delta(Y)}$ is the reward of selecting $\delta(Y)$ and $\bar{F}^*(\emptyset, Z) = 0$. The optimal decision variable for each state (Y, Z) is denoted by $\mathbf{y}^*(Y, Z)$.

Example (continued): In the previous example, the pricing problem corresponding to test 3 is denoted by PR$[N_3, \varsigma_3, \beta_3]$, where $N_3 = \{1, 2, 4\}$, $B_3 = \emptyset$, $c_3 = 40$ Let us illustrate the calculation with $\beta_3 = \{-10, -20, 0, -20\}$. Figure 4 presents the hierarchy of Ω. As shown in the figure, all the terminal nodes are singleton for which the corresponding objective value can be obtained via (5.20). With a backward recursion, this value can then be propagated upward in the tree of supproblems. Using any ordering $\preceq_\Omega$ satisfying $(Y_1, Z_1) \preceq (Y_2, Z_2) \Leftrightarrow |Y_1| \leq |Y_2|$, computing $\bar{F}(Y, Z)$ only depends on previously computed objective values. Table 1 presents these values for all the states. Based on the table, an optimal solution to this example is $\mathbf{y}^*(N_3, B_3) = (1, 1, 0, 0)$ with $\bar{F}^*(N_3, B_3) = -6.4$. □

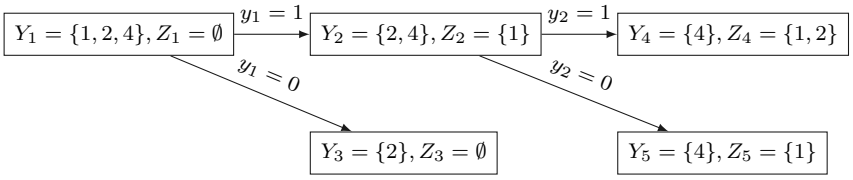


Figure 4: The state space of the example

State Index	$R(Y,Z)$	$F^*(Y,Z)$	$\mathbf{y}^*(Y,Z)$
5	-12.4	-12.4	1
4	3.8	0	0
3	0	0	0
2	2	-12.4	1
1	6	-6.4	1

Table 1: The calculation details of the example

5.1.2 A relaxation of PR

In this section, we show that by solving a relaxation of PR, we can solve the pricing problem more efficiently. We relax the multiplicative part of the objective (5.17) by replacing $F(Y, \varsigma, b; \mathbf{y})$ with $F'(Y, b; \mathbf{y}) = \sum_{j \in Y}(-b_j)y_j$. We denote this relaxation by RPR. Each instance of RPR can be converted into a special case of the maximal closure problem [29] by reversing the precedence arcs of its corresponding graph. Picard [29] shows that the maximal closure of a graph can be found by solving a maximal flow problem, and hence, can be solved very efficiently [19].

Below, we prove that solving RPR to optimality can reduce the size of PR. In order to employ the set operators, in the following, we use the set representation $C \subset N$ of the solutions to PR (recall $C_i^m \subset N$), rather than their vector representation $\mathbf{y}$. For each $i \in C$ we have $y_i = 1$. We also assume that $F(C) \equiv F(\mathbf{y})$.

Lemma 1. *If $C_1, C_2 \subseteq Y$ are two feasible solutions to* PR$[Y, \varsigma, b]$*, then $(C_1 \cup C_2) \subseteq Y$ is also a feasible solution to* PR$[Y, \varsigma, b]$.

Proof: A solution $C \subseteq Y$ is feasible iff for all $j \in C : B_j \subset C$. Moreover, we know that for any $j \in C_1 \cup C_2$, j belongs to C_1 and/or C_2. In either case, for any $i \in C_1$ we have $B_j \subset C_1$, and consequently $B_j \subset C_1 \cup C_2$. The same holds for any $j \in C_2$. Thus, we conclude that $\forall j \in C_1 \cup C_2 : B_j \subset C_1 \cup C_2$, and consequently, $C_1 \cup C_2$ is a feasible solution. □

Similarly, we can prove:

Lemma 2. *If $C_1, C_2 \subseteq Y$ are two feasible solutions to* PR$[Y, \varsigma, b]$*, then* $(C_1 \cap C_2) \subseteq Y$ *is also a feasible solution to* PR$[Y, \varsigma, b]$.

Using these two lemmas, we prove:

Theorem 5. *If C is an optimal solution to* RPR$[Y, b]$*, then there exists an optimal solution C^* to* PR$[Y, \varsigma, b]$ *such that $C \subseteq C^*$.*

Proof: We define $\bar{C} = C \setminus C^*$. Assuming $\bar{C} \neq \emptyset$ implies that $\exists j \in C : j \notin C^*$. We then have $F(C^* \cup \bar{C}) = \varsigma \prod_{j \in (C^* \cup \bar{C})} p_j - \sum_{j \in (C^* \cup \bar{C})} b_j$. Regarding the multiplicative part of the equation, since $\varsigma \geq 0$, we have $\varsigma \prod_{j \in C^*} p_j \geq \varsigma \prod_{j \in (C^* \cup \bar{C})} p_j$. With respect to the additive part, we consider the two following cases. If $\sum_{j \in \bar{C}} b_j > 0$, then based on Lemma 1 and the fact that $C^* \cup \bar{C} = C^* \cup C$, we have $F(C^* \cup \bar{C}) < F(C^*)$, which contradicts the optimality of C^* in PR. If $\sum_{j \in \bar{C}} b_j < 0$, on the other hand, then considering Lemma 2 and that $C \setminus \bar{C} = C^* \cap C$, we know $F'(C \setminus \bar{C}) < F'(C)$, which contradicts the optimality of C in RPR. We conclude that $\bar{C} = \emptyset$ and $C \subseteq C^*$. Remark that $\bar{C} \neq \emptyset$ only if $\forall j \in \bar{C} : p_j = 1$ and $\sum_{j \in \bar{C}} b_j = 0$, in which case $F(C^* \cup C) = F(C^*)$. □

Therefore, given an optimal solution C to RPR$[Y, b]$ we can replace PR$[Y, \varsigma, b]$ by PR$[Y \setminus C, \varsigma \prod_{j \in C} p_j, b]$. Moreover, if $F(C) < \alpha_i$, we can add the corresponding column to RLF without actually solving PR explicitly. Throughout the implementation of the algorithms, we use Dinic's algorithm [16] with complexity $\mathcal{O}(n^2|E|)$ to solve RPR.

5.2 Branch-and-price

The output of the B&P algorithm is an optimal integral solution to ILF. It starts by obtaining an optimal fractional solution to LF in the root node. Branching on the original variables x_{ij} rather than the pricing variables z_i^m is known to lead to more balanced search trees, and to preserve the tractability of the pricing problems [35]. Therefore, at each node, we fix a fractional variable x_{ij} to either 0 or 1. This yields two subproblems with additional constraints that can be solved by CG. The branching decisions should be imposed to PR, i.e., a variable that is fixed to zero should not be selected as entering variable in PR. To do so, we modify the precedence network, i.e., for any $\{i, j\} \in N$, fixing $x_{ij} = 0$ adds (j, i) and its transitive constraints to E. The additional constraints may empty the solution space of the restricted master problem, in which case new feasible heuristic solutions (see Section 6.2) are used to add new columns.

6 Computational results

6.1 Experimental setup

All the computational experiments are performed on a Dell Latitude E7250 laptop with an Intel Core i5-5200U (2.20GHz) processor. A memory limit of 8GB and a time limit of two hours are applied. The algorithms are coded in C++. For solving the linear programs, we use CPLEX version 12.6.3.

We use the dataset of [39], which was generated by the random network generator RanGen [15], and which is available online[1]. It contains 1080 instances in 108 different settings. Each setting is defined by a probability interval PI $\in \{l, m, h\}$, an order strength value OS $\in \{0.4, 0.6, 0.8\}$ and $n \in \{10, 20, \ldots, 120\}$. PI values represent low, medium and high success probabilities as follows: l means $p_i \in [0, 0.2]$, m implies $p_i \in [0.2, 0.8]$, and h indicates $p_i \in [0.8, 1]$. The success probability of each test is chosen randomly from the corresponding interval. OS is a measure of the density of a precedence network and is defined as the number of precedence-related test pairs divided by the maximum possible number of such pairs. The cost of each test is chosen randomly from $\{0, 1, \ldots, 50\}$. Each instance is identified by (PI, OS, n, k), where $k \in \{1, 2, \ldots, 10\}$ is the index of the instance in the setting. The three instances $(l, 0.4, 90, 1)$, $(m, 0.4, 90, 1)$ and $(h, 0.4, 90, 1)$ include identical precedence networks and testing costs, but have different success probabilities.

6.2 Implementation details

In both DPn and DPp, we use binary search to find existing states in Φ and Ω. In implementing DPp, and with minor modifications in PR, we define the states by $(Y, \prod_{j \in Z} p_j)$ rather than (Y, Z). This alternative offers a more compact representation of Ω, as any two states $(Y, Z), (Y, Z') \in \Omega$ with $\prod_{j \in Z} p_j = \prod_{j \in Z'} p_j$ will map to the same objective value in $\bar{F}()$. In B&P, we have experimented with different branching orders including the lexicographical order, the fractional variable closest to 0.5 first, the fractional variable corresponding to activities with most successors/predecessors first, etc. We observe that the lexicographical order performs the best.

We use a *Greedy Randomized Adaptive Search Procedure* (GRASP) [18] for finding high-quality sets of initial columns for CG. The procedure iteratively produces heuristic sequences and keeps track of the best produced solution. Each individual is constructed by gradually

[1] https://feb.kuleuven.be/public/u0004371/system_testing.htm

appending eligible tests to the sequence based on biased random sampling, where an eligible test with lower ratio $c_i/(1-p_i)$ has a higher selection probability.

6.3 Comparing the results

Table 2 compares the performance of our DP algorithm (DPn) with B&P and the DP algorithm of Wei et al. [39] (DPk), which is the current state of the art. The comparison is performed with and without using the precedence theorems (TH) for strengthening the networks. The column TH+DPn, for instance, shows the performance of DPn after tightening the precedence networks via the precedence theorems. Note that DPk was developed for k:n systems, but in our computations we set $k = n$. The number of instances in each (OS, n) setting (30 instances) that are solved to optimality is labeled as '#'. The average runtimes (CPU) over the solved instances are expressed in seconds. For the DP algorithms, all the unsolved instances are due to memory insufficiency, while the bottleneck of B&P for all the unsolved instances is the time limit.

Based on the table, we observe that for all three algorithms, the precedence theorems clearly have a strongly beneficial effect. TH+DPn has the best performance among all the six methods. It increases the size of solvable instances by up to 100% for OS = 0.4 when compared to DPk. In this setting, 29 out of 30 of the largest instances (with 120 tests) are solved to optimality. For OS = 0.6, for the very first time in the literature, all the instances of the dataset are solved. The largest state space that is solved by DPn without precedence theorems corresponds to an instance with OS = 0.4 and $n = 90$, which has about 3.5×10^9 states. This is 245 times larger than the largest state space that can be processed via DPk (OS = 0.4, $n = 60$).

With respect to runtimes, DPn is faster than DPk, and TH+DPn performs better than TH+DPk. Overall, TH+DPn outperforms DPk by a factor of more than 200. These improvements are the results of (1) increasing the density of the precedence networks via Theorems 1, 2 and 3, (2) eliminating the explicit enumeration of UDCs in our memory management, and (3) using Theorem 4 to eliminate the unnecessary checks of the memory management technique of [31]. Note that TH+DPk outperforms DPn, which indicates that the beneficial effect of the precedence theorems is significantly stronger than the improving effect of the new memory management technique.

The size of instances solvable by B&P is limited to 80 tests for OS = 0.8, to 70 tests for OS = 0.6, and to 50 tests for OS = 0.4. Further details on the computational results of B&P are presented in Table 3, where we report the total number of nodes in the B&B tree (node)

OS	n	TH+DPn		DPn		TH+DPk		DPk		TH+B&P		B&P	
		CPU	#	CPU	#	CPU	#	CPU	#	CPU	#	CPU	#
0.8	10	0.00	30	0.00	30	0.00	30	0.00	30	0.01	30	0.06	30
	20	0.00	30	0.00	30	0.00	30	0.00	30	0.06	30	0.38	30
	30	0.00	30	0.00	30	0.00	30	0.00	30	0.34	30	1.39	30
	40	0.00	30	0.00	30	0.00	30	0.00	30	1.06	30	10.19	30
	50	0.00	30	0.00	30	0.00	30	0.00	30	3.08	30	35.24	30
	60	0.00	30	0.00	30	0.00	30	0.01	30	6.72	30	852.34	30
	70	0.00	30	0.00	30	0.01	30	0.01	30	21.56	29	—	0
	80	0.00	30	0.00	30	0.01	30	0.02	30	345.56	14	—	0
	90	0.00	30	0.01	30	0.02	30	0.05	30	—	0	—	0
	100	0.00	30	0.02	30	0.02	30	0.09	30	—	0	—	0
	110	0.00	30	0.05	30	0.03	30	0.22	30	—	0	—	0
	120	0.00	30	0.11	30	0.05	30	0.45	30	—	0	—	0
0.6	10	0.00	30	0.00	30	0.00	30	0.00	30	0.02	30	0.23	30
	20	0.00	30	0.00	30	0.00	30	0.00	30	0.26	30	1.59	30
	30	0.00	30	0.00	30	0.00	30	0.00	30	1.27	30	226.64	30
	40	0.00	30	0.00	30	0.00	30	0.01	30	18.09	30	659.79	24
	50	0.00	30	0.02	30	0.00	30	0.04	30	741.22	30	6235.40	10
	60	0.00	30	0.07	30	0.02	30	0.24	30	3805.90	21	—	0
	70	0.01	30	0.31	30	0.04	30	1.12	30	4023.52	12	—	0
	80	0.03	30	1.19	30	0.15	30	4.11	30	—	0	—	0
	90	0.08	30	4.96	30	0.33	30	15.57	30	—	0	—	0
	100	0.20	30	23.79	30	0.77	30	61.55	27	—	0	—	0
	110	0.59	30	112.14	30	2.76	30	—	0	—	0	—	0
	120	1.57	30	579.20	30	5.22	30	—	0	—	0	—	0
0.4	10	0.00	30	0.00	30	0.00	30	0.00	30	0.04	30	1.49	30
	20	0.00	30	0.00	30	0.00	30	0.00	30	0.69	30	42.18	30
	30	0.00	30	0.01	30	0.00	30	0.03	30	79.23	29	2474.86	20
	40	0.00	30	0.13	30	0.00	30	0.27	30	1099.44	27	1972.60	10
	50	0.02	30	0.90	30	0.05	30	2.51	30	3770.86	9	—	0
	60	0.08	30	11.58	30	0.19	30	31.61	30	—	0	—	0
	70	0.49	30	89.33	30	0.95	30	—	0	—	0	—	0
	80	2.05	30	877.90	30	5.10	30	—	0	—	0	—	0
	90	12.58	30	8705.59	30	39.67	30	—	0	—	0	—	0
	100	49.88	30	—	0	204.15	30	—	0	—	0	—	0
	110	279.51	30	—	0	688.24	12	—	0	—	0	—	0
	120	921.00	29	—	0	—	0	—	0	—	0	—	0

Table 2: Computational results of the three algorithms

and the LP integrality gaps (%LP). The results of the table are visualized in Figure 5. As expected, runtimes increase as n increases or OS decreases. The results indicate that the quality of the LP bound is a major restriction. While the bound is tight when PI $= h$, its quality deteriorates when PI $\in \{l, m\}$. Consequently, the number of branched nodes increases in the latter two settings. Another observation is that the average runtime of the instances with PI $= h$ and $n = 60$ is higher than the average runtime of the instances with the same size but PI $\in \{l, m\}$. This implies that the branching tree is smaller when PI $= h$, but obtaining an optimal fractional solution at each node is more time-consuming. By profiling our code, we also observe that the percentage of the B&P runtime that is spent on pricing increases as the OS decreases, the length of the probability interval increases, or

PI	n	OS = 0.8				OS = 0.6				OS = 0.4			
		CPU	node	%LP	#	CPU	node	%LP	#	CPU	node	%LP	#
l	10	0.13	4.20	5.23	10	0.51	19.00	53.15	10	3.12	155.80	76.57	10
	20	0.61	18.00	22.37	10	2.40	90.80	65.18	10	67.85	2836.20	92.56	10
	30	1.55	29.40	56.11	10	289.56	10305.60	81.22	10	3550.10	84750.86	99.89	7
	40	16.50	489.60	65.78	10	691.52	13778.50	98.65	8	—	—	—	0
	50	54.07	1080.00	84.86	10	—	—	—	0	—	—	—	0
	60	49.60	3646.80	83.13	10	—	—	—	0	—	—	—	0
m	10	0.03	0.60	0.34	10	0.15	6.20	6.74	10	1.30	82.60	31.03	10
	20	0.40	13.00	5.02	10	2.10	84.80	13.60	10	58.19	2240.80	52.82	10
	30	1.81	50.60	10.71	10	386.45	11006.40	46.80	10	7727.23	41908.00	84.68	3
	40	11.57	335.00	34.96	10	1178.54	9683.67	59.82	6	—	—	—	0
	50	45.68	813.00	36.62	10	—	—	—	0	—	—	—	0
	60	150.41	2308.40	37.96	10	—	—	—	0	—	—	—	0
h	10	0.02	0.00	0.00	10	0.04	0.20	0.02	10	0.05	0.80	0.05	10
	20	0.14	0.00	0.00	10	0.25	0.20	0.00	10	0.51	1.40	0.08	10
	30	0.80	3.40	0.03	10	3.92	3.40	0.15	10	146.48	208.80	1.37	10
	40	2.50	1.20	0.10	10	323.16	14.20	0.11	10	1972.60	702.20	1.02	10
	50	5.98	7.20	0.01	10	6235.40	402.20	1.10	10	—	—	—	0
	60	2357.00	36.20	0.12	10	—	—	—	0	—	—	—	0

Table 3: Detailed computational results for the B&P algorithm

the success probabilities decrease. While this percentage is consistently negligible (less than 1%) when PI = h or OS = 0.8, it increases up to 30% for PI = m and OS = 0.4.

7 Conclusions

In this article, we have studied the sequential testing problem for n:n systems. We have developed a linear formulation for the problem via Dantzig-Wolfe decomposition of a compact nonlinear formulation. We have proved three precedence theorems that can identify dominant precedence constraints, which are respected by at least one optimal solution. The theorems can be used in a pre-processing step to increase the density of the precedence networks.

A DP algorithm has been proposed for solving the problem. Using a novel memory management technique, our DP significantly reduces the memory requirements and the computation times. We have empirically shown that the combination of the precedence theorems and our DP is faster than the current best performing procedure by a factor of more than 200, and increases the size of instances solvable within practical runtimes by up to 100%.

For comparison purposes, a B&P algorithm has also been devised for solving the integer linear formulation. We have shown that solving a relaxation of the pricing problem can improve the efficiency of its solution method. This relaxation can be solved in polynomial time as a maximal flow problem. Our computational results indicate that the performance of B&P is inferior to the DP, mainly due to weak LP bounds.

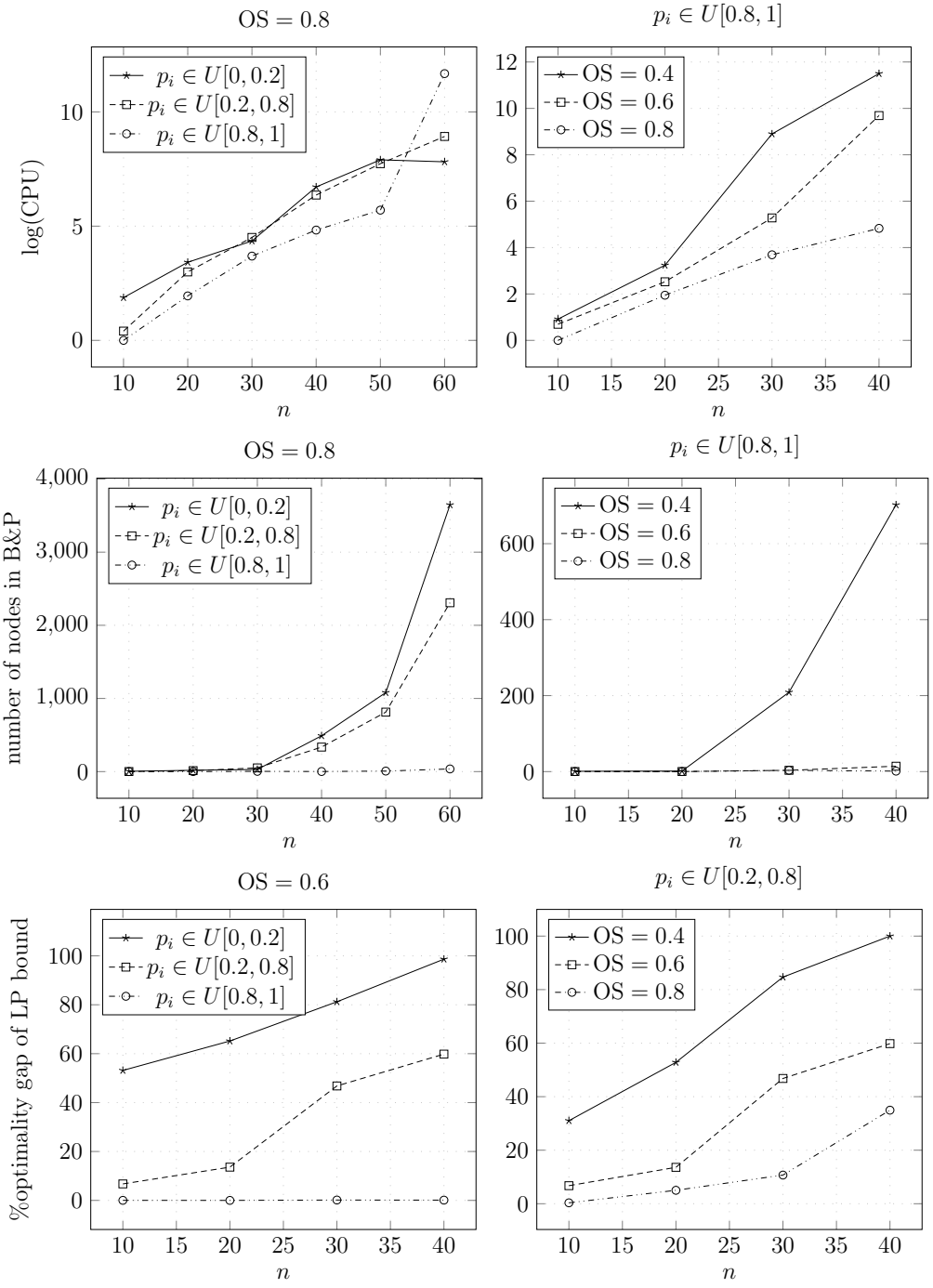


Figure 5: Computational results of B&P for different parameter settings

References

[1] U.S. Food & Drug Administration (FDA). http://www.fda.gov. Accessed: 2016-09-30.

[2] G. Bellala, S. K. Bhavnani, and C. Scott. Group-based active query selection for rapid diagnosis in time-critical situations. *IEEE Transactions on Information Theory*, 58(1): 459–478, 2012.

[3] Y. Ben-Dov. Optimal testing procedures for special structures of coherent systems. *Management Science*, 27(12):1410–1420, 1981.

[4] H. Boothroyd. Least-cost testing sequence. *Journal of the Operational Research Society*, 11(3):137–138, 1960.

[5] E. Boros and T. Ünlüyurt. Diagnosing double regular systems. *Annals of Mathematics and Artificial Intelligence*, 26(1-4):171–191, 1999.

[6] R. Butterworth. Some reliability fault-testing models. *Operations Research*, 20(2):335–343, 1972.

[7] M. F. Chang, W. Shi, and W. K. Fuchs. Optimal diagnosis procedures for k-out-of-n structures. *IEEE Transactions on Computers*, 39(4):559–564, 1990.

[8] S. Y. Chiu, L. A. Cox Jr, and X. Sun. Optimal sequential inspections of reliability systems subject to parallel-chain precedence constraints. *Discrete Applied Mathematics*, 96:327–336, 1999.

[9] K. Coolen, W. Wei, F. Talla Nobibon, and R. Leus. Scheduling modular projects on a bottleneck resource. *Journal of Scheduling*, 17(1):67–85, 2014.

[10] S. Creemers, R. Leus, and M. Lambrecht. Scheduling Markovian PERT networks to maximize the net present value. *Operations Research Letters*, 38(1):51–56, 2010.

[11] S. Creemers, B. De Reyck, and R. Leus. Project planning with alternative technologies in uncertain environments. *European Journal of Operational Research*, 242(2):465–476, 2015.

[12] R. Daldal, I. Gamzu, D. Segev, and T. Ünlüyurt. Approximation algorithms for sequential batch-testing of series systems. *Naval Research Logistics (NRL)*, 63(4):275–286, 2016.

[13] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[14] B. De Reyck and R. Leus. R&D-project scheduling when activities may fail. *IIE Transactions*, 40:367–384, 2008.

[15] E. Demeulemeester, M. Vanhoucke, and W. Herroelen. RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1):17–38, 2003.

[16] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Mathematics Doklady*, 11(5):1277–1280, 1970.

[17] S. O. Duffuaa and A. Raouf. An optimal sequence in multicharacteristics inspection. *Journal of Optimization Theory and Applications*, 67(1):79–86, 1990.

[18] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[19] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[20] M. R. Garey. Optimal task sequencing with precedence constraints. *Discrete Mathematics*, 4(1):37–56, 1973.

[21] B. Gluss. An optimum policy for detecting a fault in a complex system. *Operations Research*, 7(4):468–477, 1959.

[22] S. M. Johnson. Optimal sequential testing. Technical Report RM-1652, Rand Corporation, 1956.

[23] J. Kanet. New precedence theorems for one-machine weighted tardiness. *Mathematics of Operations Research*, 32(3):579–588, 2007.

[24] F. P. Kelly. A remark on search and sequencing problems. *Mathematics of Operations Research*, 7(1):154–157, 1982.

[25] D. Madigan, S. Mittal, and F. Roberts. Efficient sequential decision-making algorithms for container inspection operations. *Naval Research Logistics (NRL)*, 58(7):637–654, 2011.

[26] R. K. Martin. *Large Scale Linear and Integer Optimization: A Unified Approach*. Springer Science & Business Media, 2012.

[27] L. G. Mitten. An analytic solution to the least cost testing sequence problem. *Journal of Industrial Engineering*, 11(1):17, 1960.

[28] C. L. Monma and J. B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, 4(3):215–224, 1979.

[29] J.-C. Picard. Maximal closure of a graph and applications to combinatorial problems. *Management Science*, 22(11):1268–1272, 1976.

[30] C. N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. In *Combinatorial Optimization II*, pages 78–87. Springer, 1980.

[31] S. Rostami, S. Creemers, and R. Leus. Precedence theorems and dynamic programming for the single-machine weighted tardiness problem. Technical Report KBI_1715, KU Leuven, Faculty of Economics and Business, 2017.

[32] S. N. Salloum. *Optimal testing algorithms for symmetric coherent systems.* PhD thesis, University of Southern California, 1979.

[33] S. N. Salloum and M. A. Breuer. Fast optimal diagnosis procedures for k-out-of-n: G systems. *IEEE Transactions on Reliability*, 46(2):283–290, 1997.

[34] N. Samphaiboon and Y. Yamada. Heuristic and exact algorithms for the precedence-constrained knapsack problem. *Journal of Optimization Theory and Applications*, 105 (3):659–676, 2000.

[35] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[36] T. Ünlüyurt. Sequential testing of complex systems: A review. *Discrete Applied Mathematics*, 142:189–205, 2004.

[37] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.

[38] B. J. Wagner and D. J. Davis. Discrete sequential search with group activities. *Decision Sciences*, 32:557–573, 2001.

[39] W. Wei, K. Coolen, and R. Leus. Sequential testing policies for complex systems under precedence constraints. *Expert Systems with Applications*, 40(2):611–620, 2013.

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be