Resource allocation by means of project networks: dominance results

Roel Leus

Research group ORSTAT, Faculty of Business and Economics
Katholieke Universiteit Leuven, Belgium
Roel.Leus@econ.kuleuven.be

This paper investigates the relationship between resource allocation and ES-policies, which are a type of scheduling policies introduced for stochastic scheduling and which can be represented by a directed acyclic graph. We present a formal treatment of resource flows as a representation of resource-allocation decisions, extending the existing literature. Our results lead to suggestions for efficiency enhancements to enumeration algorithms for ES-policies.

Keywords: project scheduling, resource constraints, resource allocation.

1 Introduction

We will examine the scheduling of a single project. The project consists of a set \( N = \{0, 1, \ldots, n\} \) of activities, which are to be scheduled on a set \( K \) of renewable resource types with availability \( a_k, k \in K \) (e.g., groups of equivalent workers or machines). Each activity \( i \) has a fixed duration \( d_i \in \mathbb{N} \) and occupies a fixed number \( r_{ik} \in \mathbb{N} \) of units of each resource type \( k \) during its execution. The (dummy) activities 0 and \( n \) have zero duration and zero resource usage; we assume that \( d_i > 0 \) for \( i \neq 0, n \). The quantity \( s_i \geq 0 \) represents the starting time of activity \( i \); the starting-time vector \( s = (s_0, s_1, \ldots, s_n) \) is a schedule.

In most projects, some of the activities can only be started once other activities have been completed. These precedence relationships can be represented by a precedence graph; Figure 1(a) (borrowed from [16]) represents such a precedence network for a small project with six activities, so \( N = \{0, 1, \ldots, 5\} \). Throughout the article, we adopt the activity-on-the-node representation, in which each activity corresponds with a node in the network and in which the edges represent precedence constraints. The arc \((2, 3)\), for instance, indicates that activity 3 can be started no sooner than after activity 2 has ended, or in other words that \( s_2 + d_2 \leq s_3 \). Activities 0
Figure 1: Example project network and activity data.

and 5 represent the start and the completion of the project: activity 0 is a predecessor, whereas activity 5 is a successor, of all other activities. In our example project, the resource availability of a single resource type ($|K| = 1$) is $a_1 = 3$ units. All remaining data are provided in Figure 1(b) (we write $r_{1i}$ as $r_i$). A possible schedule for this project is graphically represented in Figure 2(a). This schedule does not yet contain detailed information about which of the individual resource units that make up the resource type are actually assigned to each activity. Two possible such allocations are depicted in Figures 2(b) and 2(c), where each horizontal band corresponds with (e.g.) one worker (these resource units are indicated in the figures as $w_i$, $i = 1, 2, 3$).

The goal of this article is to investigate the foregoing detailed resource allocation. To this aim, we will use transshipment networks that represent...
the flow of resource units between activities; these networks are subsequently referred to as *(resource) flow networks*. The resource flow networks corresponding with Figures 2(b) and 2(c) are depicted in Figures 3(a) and 3(b). The dummy activities 0 and 5 function as source and sink for the three resource units of the single resource type: the three units are dispatched into the network from activity 0 and gathered at node 5. Obviously, if more than one resource type is considered (\(|K| > 1\)), there will be a separate flow network for each resource type.

In the flow networks, some resource units are transported between activities that are not originally precedence-related (e.g., between activities 1 and 4). If we decide to maintain the same resource allocation throughout the execution of the project then arcs such as \((4, 1)\) in the flow network induce additional ‘hard’ precedence constraints. In fact, once a decision has been made regarding the allocation of resources and as long as all (original and extra) precedence constraints are respected, we can disregard resource constraints altogether and still produce a resource-feasible schedule. The schedule in Figure 2(a), for instance, is the result of starting all activities as early as possible subject to the original precedence constraints augmented with the extra arcs from either Figure 3(a) or 3(b).

In this article, we also investigate so-called *ES-policies* (*early-start policies*), whose purpose is to include extra arcs in the precedence network in order to guarantee that resource conflicts can no longer occur. It is clear that flow networks and ES-policies are related. The contributions of this paper are twofold: (1) we examine the relationship between resource flows and ES-policies (in Section 3), and (2) we provide suggestions for efficiency enhancements to enumeration algorithms for ES-policies (Section 4). Definitions and a literature review are given in Section 2. We briefly summarize our findings in Section 5.

![Flow networks](image)

Figure 3: Flow networks corresponding with the resource allocations in Figures 2(b) and 2(c). Flow quantities are indicated next to each arc.
2 Definitions and literature review

In this section, we present a number of general definitions on project scheduling (Section 2.1), we formally introduce the concepts of resource flows (Section 2.2) and of ES-policies (2.3), we link our work with the area of network design (Section 2.4) and we discuss possible objective functions (Section 2.5).

2.1 Project scheduling

We gather all technological precedence constraints in a binary relation $A \subset N \times N$. We assume that $A$ is a (strict) partial order on $N$, i.e. an irreflexive and transitive relation; henceforth we will call $A$ a precedence relation. The dummy activities 0 and $n$ are the unique least and greatest element of the partially ordered set $(N, A)$, so for all $i \in N \setminus \{0\}$: $(0, i) \in A$, and for all $i \in N \setminus \{n\}$: $(i, n) \in A$. If $(i, j) \in A$ then we require that $s_i + d_i \leq s_j$ (which are the classic finish-start precedence constraints with zero time-lag).

For a binary relation $E$ on $N$, let $T(E)$ denote its transitive closure, defined as the minimal transitive relation on $N$ that contains $E$. The transitive reduction $t(E)$ of a binary relation $E$ on a set $N$ is the minimal relation on $N$ such that $T(t(E)) = T(E)$. Informally, $t(E)$ contains only the ‘direct’ arcs in the precedence relation, whereas $T(E)$ contains all possible direct as well as implicit arcs. Notice that it is customary to include only direct activity pairs in $t(A)$ in the precedence network. For the precedence network shown in Figure 1(a), for instance, the actual precedence relation $A$ also contains the elements $(0, 3), (0, 5)$ and $(2, 5)$, which are not included in the network. The transitive reduction of a partial order is unique [1], and so is its transitive extension.

For a binary relation $E$ on $N$, we define $S(E) = \{s \in \mathbb{R}_{\geq}^{n+1} : s_i + d_i \leq s_j, \forall (i,j) \in E\}$, where $\mathbb{R}_{\geq}$ denotes the set of non-negative real numbers. If $E$ is a precedence relation then $S(E)$ is the set of starting-time vectors that respect all the constraints imposed by $E$. Set $S(E)$ is non-empty if and only if the corresponding graph $G(N, E)$ is acyclic. The set of time-feasible schedules is $S(A)$. Clearly, if $A \subseteq E$ then $S(E) \subseteq S(A)$.

Without loss of generality, we restrict our attention to integer components for the vector $s$. The time interval $[t-1, t]$ is referred to as (time) period $t$, for integer $t$. For period $t \in N_0$, we define the set $A(s, t) = \{i \in N : s_i \leq (t - 1) \land s_i + d_i \geq t\}$, containing the activities that are in progress (or ‘active’) during period $t$ in schedule $s$. A schedule $s$ is called feasible if it is
both time-feasible and resource-feasible; the latter property entails
\[ \sum_{i \in A(s,t)} r_{ik} \leq a_k, \quad \forall t \in \mathbb{N}_0, \forall k \in K. \]

The Resource-Constrained Project-Scheduling Problem (RCPSP) aims at finding a feasible schedule \( s \) that minimizes a cost function, usually \( s_n \) (see, for example, [7, 24]). For more information on the objective function, see Section 2.5.

### 2.2 Resource flows

The use of resource flows has been advanced by various authors, among whom [2, 6, 16, 23, 24]. As early as 1964, the main idea underlying the concept of resource flows was used by Wiest [31] in his ‘critical sequence’, which is a chain of either precedence-related or resource-related activities that determines the length of the project. In this article, the word flow mostly refers to a resource flow. A flow \( f \) assigns to each triple \((i, j, k) \in N \times N \times K\) a value \( f(i, j, k) \in \mathbb{N} \), which represents the number of resource units of type \( k \) that are transferred from activity \( i \) (when it finishes) to activity \( j \) (when it starts). These values must satisfy the following constraints, which constitute flow-conservation constraints and lower and upper bounds on the flow through intermediate (non-dummy) nodes:

\[
\sum_{j \in N} f(j, i, k) = \sum_{j \in N} f(i, j, k) = r_{ik}, \quad \forall i \in N \setminus \{0, n\}, \forall k \in K. \quad (1)
\]

A flow \( f \) corresponds with one flow network per resource type \( k \in K \). For each type \( k \), a number \( a_k \) of resource units (availability of the resource type) is sent into the network from the dummy start node and collected at the end node:

\[
\sum_{j \in N} f(0, j, k) = \sum_{j \in N} f(j, n, k) = a_k, \quad \forall k \in K. \quad (2)
\]

Remark that we are not dealing with multi-commodity flows: each resource type has its own distinct capacity. The resource flow networks send flow across a subset of the elements of \( N \times N \); for a flow \( f \) we define the set of activity pairs \( \phi(f) = \{(i, j) \in N \times N : f(i, j, k) > 0 \text{ for at least one } k \in K\} \), containing the activity pairs that carry non-zero flow. We are most interested only in those flow-carrying arcs that are not in \( A \) and so do not coincide with technological precedence constraints; these are grouped in the set \( C(f) = \)
\( \phi(f) \setminus A \). A resource flow \( f \) entails a detailed resource-allocation decision for the individual resource units that compose a resource type and, under the condition of invariant resource allocation, induces additional precedence constraints via the elements of \( C(f) \) (see e.g. Bowers [6] for a discussion). We define a flow \( f \) as \textit{feasible} when \( G(N, A \cup C(f)) \) is acyclic, in which case the project can be implemented with the resource-allocation decisions inherent in \( f \).

### 2.3 ES-policies

Given resource requirements \( r_{ik} \) for all activities \( i \in N \), a set of activities \( F \subset N \) is a \textit{forbidden set} of a precedence relation \( E \) if it is an anti-chain of \( E \) (a stable set in graph \( G(N, E) \)) and if \( \sum_{i \in F} r_{ik} > a_k \) for at least one \( k \in K \). In other words, a forbidden set is a set of activities that are pairwise not precedence-related and that jointly require more than the available units of at least one resource type, so that they should not be scheduled in parallel. An inclusion-minimal forbidden set is called a \textit{minimal forbidden set} or MFS (see, for instance, Stork and Uetz [30]). We denote by \( \mathcal{F}(E) \) the set of MFSs for precedence relation \( E \) (the parameters of the RCPSP-instance under consideration are also arguments to \( \mathcal{F}() \) but are omitted). For the example project described in the Introduction, we have \( \mathcal{F}(A) = \{\{1, 2, 4\}, \{3, 4\}\} \).

In the context of scheduling under uncertainty, different scheduling policies for projects with stochastic activity durations were presented by Igelmund and Radermacher [13] based on the concept of forbidden sets. A set of policies of interest to us is the set of \textit{ES-policies} (early-start policies). The idea is to select a set \( X \subset (N \times N) \setminus A \) and to extend the partial order \( A \) to \( A \cup X \) such that \( \mathcal{F}(T(A \cup X)) = \emptyset \). In other words, the set \( X \) is chosen such that we can ignore resource constraints if we respect the extended set of precedence constraints \( A \cup X \). In line with Balas [3], we call \( X \) a \textit{selection}; for clarity, we will sometimes also use the term \textit{ES-selection}. The policy is only feasible if \( G(N, A \cup X) \) is still acyclic. A selection \( X \) of activity pairs that leads to a feasible ES-policy is called a \textit{sufficient set} or selection. \( T(A \cup X) \) is an order relation if \( X \) is sufficient; this does not necessarily hold for \( A \cup X \).

For a sufficient ES-selection \( X \), if \( s \in S(A \cup X) \) then \( s \in S(A) \) and \( s \) is resource-feasible: only anti-chains of \( T(A \cup X) \) can be active concurrently. This is useful when activity durations are variable: the ES-policy defined by \( X \) simply computes starting times based on an early-start critical-path recursion. In deterministic scheduling, the \textit{Main Representation Theorem} of Bartusch et al. [5] establishes the equivalence between RCPSP with a regular objective function (see Section 2.5) and the search for an appropriate extension of \( A \), leading to the so-called \textit{order-theoretic} approach to scheduling.
[21, 22], in which a search for an optimal schedule is replaced by the search for an optimal partial order. The model is an extension of the disjunctive-graph representation of the classic job-shop scheduling problem [27], and appears to have been known at least since the year 1971 [3].

2.4 Network design

Network design problems deal with the optimal design of a network in order to meet a given set of specifications while minimizing total cost, cf. [17, 18]. A solution is a network flow, and the cost of this solution is a function of the arc flow values. In the related area of graph augmentation, the problem is studied of augmenting a graph to reach a given requirement (e.g., connectivity) by adding edges [11].

A lot of network design problems are special cases of the (intractable) Minimum Edge-Cost Flow Problem (MECF, problem ND32 in Garey and Johnson [9]), where the goal is to find a maximum flow obeying edge capacities and flow-conservation laws, so that the cost of the edges carrying non-zero flow is minimized. In the setting of this article, we are mostly indifferent as to the quantity of flow on each edge and only distinguish between zero and non-zero flow on the arcs in $(N \times N) \setminus A$. Closely related recent sources in the literature on network design are [8, 14, 15, 28]. The major difference with each of these is the fact that we impose acyclicity (in order for the project to be executable), and also that we deal with (in the terminology of Minoux [18]) non-simultaneous flows: a network is to be designed for $|K|$ independent commodities.

2.5 Objective function

To measure the quality of the outcome of alternative project realizations, it is customary to use a cost function $\kappa : \mathbb{R}^{n+1}_{\geq 0} \rightarrow \mathbb{R}$ that maps a schedule $s$ to its performance measure $\kappa(s)$. Most of the popular deterministic-scheduling objective functions are regular, meaning that they are non-decreasing, which corresponds with the condition

$$(s_0, \ldots, s_n) \leq (s'_0, \ldots, s'_n) \Rightarrow \kappa(s_0, \ldots, s_n) \leq \kappa(s'_0, \ldots, s'_n),$$

where the first inequality should be read component-wise. Examples of regular objective functions are the makespan $s_n$, the weighted sum of completion times $\sum_{i \in N} c_i(s_i + d_i)$ (the weight $c_i \geq 0$ indicates the importance of activity $i$) and the weighted tardiness $\sum_{i \in N} c_i(s_i + d_i - \delta_i)^+$, where $\delta_i$ is a due date for activity $i$ and $b^+ = \max\{0, b\}$ (see [21, 24, 25, 29], for instance).
The objective function in stochastic scheduling is usually the expected value of a cost function, most frequently the project makespan. It turns out that exact determination of such objectives in the stochastic case is usually unrealistic [10, 20], in that it is highly unlikely that it could be done in polynomial time, and simulation is normally used, cf. [12, 16, 22, 29].

3 The relation between resource flows and ES-policies

For a feasible resource flow \( f \), the set \( X = C(f) \) is a sufficient ES-selection, which follows from the fact that the summed resource usage of an anti-chain of \( T(A \cup C(f)) \) never exceeds \( a_k \) for any \( k \in K \), since the set of all incoming arcs into elements of the anti-chain is a subset of a cut in the network \( G(N, A \cup X) \). As an illustration, for the example presented in the Introduction, \( \{(1,3),(4,1)\} \) is a sufficient set. In [16] (based on [19]), it is also shown that if \( X \) is a sufficient set then a feasible flow \( f \) exists with \( A \cup C(f) \subseteq T(A \cup X) \). The goal of this section is to distinguish a class of sufficient selections that can be useful for algorithmic purposes.

We call a sufficient selection \( X \) dominant if \( (T(A \cup X) \setminus \{(i,j)\}) \setminus A \) is not sufficient for all \( (i,j) \in t(A \cup X) \setminus A \). A selection that is not dominant imposes stricter precedence constraints than necessary and will therefore also be dominated with respect to most logical objective functions, since it corresponds with an unnecessary restriction of the solution space. For the example project presented in Section 1, we refer to the flows \( f_1 \) and \( f_2 \) as described in Figure 3. The selection \( C(f_1) = \{(4,1),(4,3)\} \) is dominant since \( F(C(f_1) \setminus \{(4,1)\}) = \{1,2,4\} \) and \( F(C(f_1) \setminus \{(4,3)\}) = \{3,4\} \). The selection \( C(f_2) = \{(1,3),(4,1)\} \), on the other hand, is not dominant because \( F((C(f_2) \cup \{(4,3)\}) \setminus \{(1,3)\}) = \emptyset \).

The following lemma is useful in what follows:

**Lemma 1.** If \( X \) is sufficient and \( X \subset Y \) and \( G(N, A \cup Y) \) is acyclic, then \( Y \) is sufficient.

**Proof:** No new anti-chains can be created by adding pairs to an order relation, and acyclicity then leads to sufficiency.

We can distinguish a condition that is weaker than the one appearing in the definition of dominance:

**Lemma 2.** For a sufficient ES-selection \( X \), if \( X \) is dominant then \( (t(A \cup X) \setminus A) \setminus \{(i,j)\} \) is not sufficient, for all \( (i,j) \in t(A \cup X) \setminus A \).
Proof: If $X$ is dominant then for all $(i, j) \in t(A \cup X) \setminus A$ it holds that $(T(A \cup X) \setminus \{(i, j)\}) \setminus A$ is not sufficient. Since $(t(A \cup X) \setminus A) \subseteq (T(A \cup X) \setminus \{(i, j)\}) \setminus A$ and with Lemma 1, the lemma can be seen to hold. □

We illustrate these definitions by means of a second example project. Consider a set of activities $N = \{0, 1, 2, 3, 4, 5\}$, with 0 and 5 the dummy start and end. There are no precedence constraints between non-dummy activities: $A = \{(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5)\}$. We consider a single resource type ($|K| = 1$) with availability $a_1 = 4$ and usage by the non-dummy activities $r_1 = r_2 = r_3 = r_4 = 2$. This project has $F(A) = \{(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)\}$. Figure 4 visualizes alternative arc selections; for ease of exposition, we omit the dummy start and end node. Selection $E_0$ in Figure 4 is not sufficient because $F(T(A \cup E_0)) = \{(2, 3, 4)\}$. Selection $E_1$ is sufficient but not dominant by (the negation of) Lemma 2: $t(A \cup E_1) \setminus A = \{(1, 2), (1, 3), (2, 4)\}$, from which $(1, 3)$ can be removed while sufficiency is maintained. A dominant selection need not be minimal (meaning ‘subset-minimal’): $E_2$ is sufficient and dominant, but not minimal.

We define an activity pair $(i, j) \in C(f)$ to be minimal with respect to a feasible resource flow $f$ if $(i, j) \in t(A \cup C(f))$, or put differently, if apart from arc $(i, j)$, no path $i \rightarrow j$ exists in graph $G(N, A \cup C(f))$; note that a minimal arc is not in $A$. A feasible flow $f$ is called dominant if $f$ has no minimal activity pair $(i, j)$ such that a feasible flow $f^*$ exists with $C(f^*) \subseteq T(A \cup C(f)) \setminus \{(i, j)\}$. For the example presented in the previous paragraph, the flow $f_1$ in Figure 5 is not dominant: the minimal arcs (1, 4) and (2, 3) can be removed while the possibility of a feasible flow is preserved (e.g. $f_2$).

Theorem 1. If $X$ is a dominant ES-selection then
(1) a feasible resource flow $f$ exists with $T(A \cup C(f)) = T(A \cup X)$;

Figure 4: Arc selections for the second example; dummy nodes 0 and 5 are not depicted.
(2) each such flow is dominant;
(3) no feasible flow \( f \) exists with \( T(A \cup C(f)) \not\subseteq T(A \cup X) \).

**Proof:** Consider a dominant selection \( X \). We know that a feasible flow \( f \) exists with \( A \cup C(f) \subseteq T(A \cup X) \). By the definition of dominance, it holds that \( (T(A \cup X) \setminus \{i, j\}) \setminus A \) is not sufficient, for all \( (i, j) \in t(A \cup X) \setminus A \). As \( G(N, A \cup X) \) is acyclic, no feasible flow \( f^* \) exists with \( A \cup C(f^*) \subseteq T(A \cup ((T(A \cup X) \setminus \{i, j\}) \setminus A)) = T(A \cup X) \setminus \{i, j\} \). From this, we conclude that all \( (i, j) \in t(A \cup X) \setminus A \) are in \( C(f) \) for each feasible flow \( f \). Therefore \( T(A \cup C(f)) = T(A \cup X) \), and no feasible flow exists with \( T(A \cup C(f)) \subset T(A \cup X) \) for which the inclusion is proper.

Since \( A \cup C(f) \) and \( A \cup X \) have the same transitive extension, they also have the same transitive reduction. With respect to \( f \), consider any minimal activity pair \( (i, j) \); we see that \( (i, j) \in t(A \cup X) \setminus A \), and we showed above that no feasible flow \( f^* \) exists with \( A \cup C(f^*) \subseteq T(A \cup X) \setminus \{i, j\} \). Since \( T(A \cup X) = T(A \cup C(f)) \), \( f \) is dominant.

**Theorem 2.** If \( f \) is a dominant resource flow then \( C(f) \) is a dominant ES-selection.

**Proof:** Since \( f \) is feasible, \( X = C(f) \) is a sufficient set. Since \( f \) is dominant, it holds that no feasible flow \( f^* \) exists with \( C(f^*) \subset T(A \cup X) \setminus \{(i, j)\} \), for all \( (i, j) \in t(A \cup X) \setminus \{(i, j)\} \). Therefore, \( (T(A \cup X) \setminus \{(i, j)\}) \setminus A \) is not sufficient, and so \( X \) is dominant.

We now include the requirement of subset-minimality in the analysis.

**Lemma 3.** A dominant ES-selection \( X \) is subset-minimal if and only if \( X = t(A \cup X) \setminus A \).
A sufficient selection $X$ is dominant if $(T(A \cup X) \setminus \{i, j\}) \setminus A$ is not sufficient, for all $(i, j) \in t(A \cup X) \setminus A$. If $X = t(A \cup X) \setminus A$ then $X$ is subset-minimal because removing any of the elements of $X$ would cause $X$ to not be sufficient (by Lemma 2).

Suppose now that $X$ is minimal and that $X \neq X^* = t(A \cup X) \setminus A$. Since $t(A \cup X) \setminus A = t(A \cup X^*) \setminus A$, it follows that $X^*$ also dominant, and we can see that $X^* \subseteq X$. Therefore $X^* = X$ because strict inclusion would contradict the minimality of $X$. □

Theorem 3. A subset-minimal dominant ES-selection $X$ is exactly the set of minimal arcs of any feasible resource flow on the network $G(N, T(A \cup X))$.

Proof: Theorem 1 states that if the ES-selection $X$ is dominant then one or more feasible resource flows $f$ exist with $T(A \cup C(f)) = T(A \cup X)$, and none with $T(A \cup C(f)) \subseteq T(A \cup X)$. In the proof of Theorem 1, we showed that all $(i, j) \in t(A \cup X) \setminus A$ are in $C(f)$ for each feasible flow $f$, so that all elements of $t(A \cup X) \setminus A$, which equals $X$ by Lemma 3, are minimal arcs of $f$. Additionally, it holds that $t(A \cup C(f)) = t(A \cup X)$, from which we can conclude that there are no other remaining minimal arcs of $f$. □

This theorem allows us to partition the set of dominant ES-selections into equivalence classes with the same minimal representative. The minimal representative of selection $E_2$ in Figure 4 is $E_3 = \{(1, 2), (2, 4)\}$; a feasible flow with arcs only from $T(A \cup E_3)$ is $f_3$ with $\phi(f_3) = \{(0, 1), (0, 3), (1, 2), (2, 4), (3, 5), (4, 5)\}$ and $f_3(i, j) = 2$ for all $(i, j) \in \phi(f_3)$, where $f_3(i, j) \equiv f_3(i, j, 1)$; in this case, $f_3$ is also the only feasible flow on the network $G(N, T(A \cup E_3))$.

Lemma 4. For each arc $(i, j)$ in a subset-minimal dominant ES-selection $X$ there exists a set $F \in \mathcal{F}(T(A \cup X)) \setminus \{(i, j)\}$ such that $(i, j) \subset F$.

Proof: From Theorem 3, we know that $X$ is exactly the set of minimal arcs of any feasible resource flow $f$ on the network $G(N, T(A \cup X))$. Additionally (Theorem 1(2)), each such $f$ is dominant. Consequently, from the definition of dominant flows, for each $(i, j) \in X$ it holds that no feasible flow $f^*$ exists with $C(f^*) \subset T(A \cup C(f)) \setminus \{(i, j)\}$. Since $T(A \cup X) = T(A \cup C(f))$ (Theorem 1(1)), we see that $\mathcal{F}(T(A \cup X) \setminus \{(i, j)\})$ contains at least one MFS $F$ (because acyclicity is not an issue here). Since $\mathcal{F}(T(A \cup X)) = \emptyset$, it is the removal of $(i, j)$ that leads to the existence of $F$. Since $(i, j) \in t(A \cup X)$, we conclude that $(i, j) \subset F$. □
We say that an activity pair \((i, j) \in N \times N\) resolves an MFS \(F\) if \(\{i, j\} \subset F\). Lemma 4 states that for each arc \((i, j)\) in a minimal dominant selection \(X\), it holds that the set \(\{i, j\}\) is a subset of at least one MFS that would not be ‘resolved’ if the corresponding activity pair were removed from \(X\).

The following result is in line with [19] and motivates our definition of dominance:

**Theorem 4.** For any regular objective function, using the order-theoretic approach to scheduling, there exists an optimal solution that is a minimal dominant ES-selection.

**Proof:** If \(X\) is an optimal sufficient ES-selection that is not dominant then an activity pair \((i, j) \in t(A \cup X) \setminus A\) exists for which \(X^* = (T(A \cup X) \setminus (i, j)) \setminus A\) is also sufficient. \(X\) can thus stepwise be reduced to a dominant optimal selection.

If \(X\) is a dominant optimal ES-selection that is not minimal then a subset \(X^* \subseteq X\) exists with \(T(A \cup X^*) = T(A \cup X)\). In this way, \(X\) can be reduced to a minimal dominant optimal selection. \(\square\)

A sufficient ES-selection \(X\) need not contain arcs resolving each \(F \in \mathcal{F}(A)\), and there may be multiple ways to identify a hitting set\(^1\) \(X^* \supseteq X\) of arcs in \(T(A \cup X)\) such that each MFS is resolved by an element of \(X^*\). Theorem 4 underlines the fact that full enumeration of these sets \(X^*\) is usually unnecessary. Theorems 1, 2 and 3 describe the close relationship between network flows and ES-selections, which will be used in Section 4 to prune away part of the search tree of enumerative search procedures.

## 4 Efficient enumeration of ES-selections

As explained in Section 2.3, the order-theoretic approach to scheduling searches for an ES-selection that minimizes a performance measure of the schedule that is obtained by the corresponding ES-policy; in stochastic scheduling, this performance measure is usually the expected makespan. To the best of our knowledge, the most recent computational results on enumeration schemes for ES-policies were published in the PhD-thesis of Stork [29] at the Technische Universität Berlin in 2001; the directly relevant earlier work is

\(^1\)Consider a collection \(V\) of subsets of a set \(S\). A hitting set \(S'\) is a subset of \(S\) that contains at least one element of each subset in \(V\). For the remark made above, \(S\) consists of the arcs in \(T(A \cup X)\) and \(V\) contains one subset of \(S\) for each MFS, with all arcs resolving the MFS.
summarized by Bartusch et al. [5], Igelmund and Radermacher [12] and Radermacher [26]. Stork [29] implements an algorithm that refines the existing procedures. As a consequence, we will only refer to Stork’s algorithm below as the starting point for further efficiency enhancements, under the name MFS-branching procedure. Throughout this section, we will use the same scheduling example from [16] to illustrate the discussion. We consider the set of activities $N = \{0, 1, 2, 3, 4, 5\}$, with 0 and 5 the dummy start and end. The set of MFSs is $F(A) = \{\{1, 2, 3\}, \{1, 2, 4\}\}$. Stork and Uetz [30], in line with Radermacher [26], note that for most purposes this system of MFSs, together with the precedence constraints, constitutes the essential information that characterizes an instance of a resource-constrained scheduling problem. We assume that there are no precedence constraints between non-dummy activities. Knowledge of the exact resource availability and requirements is not necessary for this illustration; an example of specific choices that correspond with $F(A)$ above is $|K| = 1$, $a_1 = 4$, $r_1 = r_2 = 2$ and $r_3 = r_4 = 1$.

4.1 The MFS-branching procedure

The MFS-branching procedure [29] proceeds as follows. The MFSs are first ordered; suppose the sequence $(F_1, F_2, \ldots, F_{|F(A)|})$ represents the chosen complete order. Subsequently, an enumeration tree is set up. Each node $v$ in this tree is associated with a MFS $F$ and branching on $v$ systematically resolves $F$ by the creation of a child node of $v$ for each ordered pair $(i, j)$ with $\{i, j\} \subset F$. In this way, each leaf $u$ of the search tree represents an ES-policy defined by the ES-selection containing the activity pairs encountered in the nodes on the path from the root of the tree to $u$. As soon as a (possibly partial) selection $X$ leads to a cyclic graph $G(N, A \cup X)$, the node is discarded. A cyclicity check can be implemented efficiently, for instance by using a distance matrix (see Bartusch et al. [5]). Maintenance of the transitive closure $G(N, T(A \cup X))$ when exploring new branches can be embedded in the distance updates and does not add to the $O(n^2)$ time complexity of these updates.

The search tree explored by the MFS-branching procedure for the example project is depicted in Figure 6. The MFSs are ordered as follows: $F_1 = \{1, 2, 3\}$, $F_2 = \{1, 2, 4\}$. At the first level of the search tree we find the six nodes numbered (1) to (6), which represent the six possibilities for resolving the forbidden set $F_1$ (by adding the indicated arc to the precedence network). At the second level, all options for resolving $F_2$ are listed. The following direct-selection dominance rule is straightforward: the child node (7) where (1, 2) is selected for $F_2$ dominates all other child nodes of (1), which is why only (7) is depicted in the graph. Four other choices (namely (1, 4), (4, 1),
Figure 6: A part of the branching tree of the MFS-branching procedure.

(2,4) and (4,2) are dominated, and the choice (2,1) leads to a cyclic graph.

We explained in Section 2.5 that the only practical means to evaluate the objective function in stochastic scheduling is simulation, which is unfortunately quite time-consuming and makes up the bulk of the running time of enumeration algorithms. Consequently, the computational efficiency of an algorithm is directly determined by the number of unpruned leaf nodes in the search tree. The tree in Figure 6 contains a number of leaf nodes that are dominated according to Theorem 4 but are not recognized as such by the search procedure. This is the case with nodes (9) and (10): if we add (1,2) or (2,1) then retroactively it turns out that the addition of (1,3) in node (3) was not needed. In the entire search tree, eight out of the 26 explored leaf nodes are redundant.

A first adaptation that might be considered for MFS-branching procedures is to trace back for each node (leaf and other) the decisions that were made at higher levels in the tree and verify in this way whether other MFSs are implicitly already resolved. This would entail a lot of extra work, however, especially since the number of MFSs can be exponential in \( n \). We therefore propose different remedies in the next two subsections.

4.2 An alternative branching strategy: binary branching

A weakness of the MFS-branching procedure is the fact that different children of the same node in the search tree do not necessarily represent mutually exclusive decisions. In the enumeration tree of Figure 6, for instance, edge (1,2) is added in node (1) and also in a child node of node (3). Each ES-selection \( X \) that is redundant according to Theorem 4 can be associated to at least one minimal dominant representative \( X' \subseteq T(A \cup X) \setminus A \). Viewed in
this manner, the MFS-branching procedure does not examine a partition of its search space: some solutions are implicitly visited multiple times (such as dominant selection \{(1, 2)\} in the example). In general, this is an undesirable aspect of enumerative search procedures because of the obvious redundancy in computational effort and the inefficacy of pruning by bound; see Balas and Toth [4], for instance, for some similar considerations regarding branch-and-bound methods for the TSP.

An alternative branching strategy can be devised that partially does away with the foregoing problem; we refer to this strategy as binary branching. At each level of the enumeration tree, two children are now created: for one specific activity pair \((i, j)\) we investigate the disjunction

\[
(i, j) \in G(N, T(A \cup X)) \quad \lor \quad (i, j) \notin G(N, T(A \cup X)),
\]

where the ES-selection \(X\) is the requested output of the search procedure. This is illustrated in Figure 7, where the first alternative is simply written as ‘\((i, j)\)’ and the second as ‘\((i, j)\)’, for brevity. Note that case \(\overline{(i, j)}\) means that edge \((i, j)\) cannot be directly added to \(X\) nor that it can be transitively implied from the addition of other edges.

The described binary branching procedure is similar in name to the procedure proposed by Leus and Herroelen [16], but the details are considerably different: we now branch on inclusion or exclusion of an edge from the transitive closure of the extended graph, while the earlier article [16] dealt with a different problem (resource allocation for input schedules) via branching on lower and upper bounds of flow across edges.
A little scrutiny shows that for the example project, only the 18 non-dominated solutions distinguished in the previous subsection will be visited, provided that the appropriate edges are chosen as the subject of the branching decisions in the appropriate nodes. In general, although the binary branching scheme may eliminate part of the computational effort in the search for a convenient network extension, it will not necessarily eliminate all of the redundancy. This will depend on the order in which edges are selected for branching, and there is no clear guideline for this. We refer to Figure 8 for an illustration, where node (3) is a dominated leaf node. For the example project, redundancies would arise in the MFS-branching scheme regardless of the MFS-order, while the occurrence of this redundancy is dependent of the branching choice in the binary branching scheme.

An additional advantage of binary branching is the fact that knowledge of the set of MFSs (which is exponential in the number of activities) is not indispensable: the maximum depth of the search tree is quadratic in the number of activities, and any criterion for selection of a branching edge will lead to a correct search procedure. Still, the only edges that need to be branched on are those that resolve at least one MFS (see Lemma 4). Unfortunately, it turns out that answering the question whether a given arc \((i, j)\) resolves any MFS is NP-complete, which can be shown using similar arguments as [29], where PARTITION is reduced to determining whether or not a given activity \(i \in N\) is contained in some (at least one) MFS \(F \in \mathcal{F}(A)\). In [16], there was no risk of obtaining a cyclic network, and new edges to be added were selected from a resource flow network that utilized all the edges that were not yet “forbidden” (corresponding with the right branch in our branching scheme). Verifying the existence of a feasible flow on an oriented network that may contain cycles is NP-complete, however (by reduction from HAMILTONIAN PATH), whereas this can be done efficiently when the input network is acyclic (based on network-flow techniques).
4.3 A test for dominance

Theorem 4 and the foregoing subsections indicate that a check whether a given sufficient selection $X$ is dominant or minimal is potentially a valuable tool. From the order-theoretic perspective, one would be inclined to perform such checks in exponential time: if $X$ is sufficient then for each $(i, j) \in t(A \cup X) \setminus A$, one can verify dominance by iteration over all MFSs (whose number is exponential in $n$) to see if $T(A \cup X) \setminus \{(i, j)\}$ is still sufficient; a similar test can be set up for subset-minimality.

From Lemma 3, we see that the minimality criterion for a dominant selection is efficiently checked in polynomial time. The same holds for the dominance property for a sufficient selection $X$: this requires at most $|t(A \cup X) \setminus A|$ tests for sufficiency of selections $Y$ for which we know that $G(N, A \cup Y)$ is acyclic; each such test is equivalent to the verification of whether a feasible flow exists on an acyclic network (see the discussion at the start of Section 3). This can be performed in polynomial time by means of maximum-flow computations (see [16, 19]). When this test fails for, say, $(i, j) \in t(A \cup X) \setminus A$, we know that the corresponding node in the search tree can be pruned; this description yields a dominance test that can be embedded in both branching schemes of the previous subsections.

Alternatively, we can also substitute $(T(A \cup X) \setminus \{(i, j)\}) \setminus A$ for $X$ and repeat the procedure. Stepwise, the redundant selection is thus reduced to a minimal dominant selection, in line with the procedure implied by the proof of Theorem 4. The output of the feasibility test mentioned supra is actually a feasible flow in case of a ‘yes’ answer; stepwise reducing this feasible flow to a dominant flow rather than working with the corresponding ES-selections allows for significant efficiency gains (see [16]). By Theorems 2 and 3, this dominant flow yields the dominant minimal representative referred to in Section 4.2, which provides information on the level in the search tree up to which we can backtrack upon discovery of the redundant node. For the binary branching scheme, the dominant minimal representative of a redundant node can even be informative after backtracking: the minimal edges of this network are convenient choices for new branching decisions.

5 Summary and conclusions

In this article, we have studied the relationship between resource allocation and ES-policies, which are a type of scheduling policies introduced for stochastic scheduling. We have presented a formal treatment of resource flows as a representation of resource-allocation decisions, extending the existing lit-
erature. Our results have led to suggestions for efficiency enhancements to enumeration algorithms for ES-policies.

We conclude from Section 4 that our results combine into a new dominance rule enabling the identification of non-dominant and dominant but non-minimal arc selections in polynomial time, while previously only exponential-time routines were on hand. The computational benefits of this rule will depend on a number of factors, especially on the choice of the objective function and also, in case the objective is evaluated via simulation, on the desired precision, since a higher accuracy obviously entails more samples, which therefore leads to a greater benefit obtained from pruning a node in the search tree. A third obvious determinant of the numerical usefulness of this rule is the set of algorithmic choices for the search procedure.

References


