Journal of the Operational Research Society (2005) 0, 1–10 © 2005 Operational Research Society Ltd. All rights reserved. 0160-5682/05 $30.00
www.palgrave-journals.com/jors

Minimizing sum of completion times on a single machine with sequence-dependent family setup times

S Karabatı[1]* and C Akkan[2]
[1]Koç University, Rumelifeneri Yolu, Sarıyer, İstanbul, Turkey; [2]Sabancı University, Orhanlı, Tuzla, İstanbul, Turkey

This paper presents a branch-and-bound (B&B) algorithm for minimizing the sum of completion times in a single-machine scheduling setting with sequence-dependent family setup times. The main feature of the B&B algorithm is a new lower bounding scheme that is based on a network formulation of the problem. With extensive computational tests, we demonstrate that the B&B algorithm can solve problems with up to 60 jobs and 12 families, where setup and processing times are uniformly distributed in various combinations of the [1,50] and [1,100] ranges.
Journal of the Operational Research Society (2005) **0,** 000–000. doi:10.1057/palgrave.jors.2601989

Keywords: single machine scheduling; sum of completion times; family setups; branch-and-bound; Lagrangean relaxation

Introduction

Setup times are an integral part of manufacturing operations. In a typical manufacturing operation, a setup time is incurred when jobs that have different processing requirements, such as the tooling requirement, are processed on the same resource. When jobs can be clustered into groups based on the similarity of their processing requirements, sequencing decisions become important in terms of achieving the associated efficiency gains.[1]

In this paper, we address the single-machine scheduling problem where jobs can be grouped into *families*. If the sequence requires a switch from a job in a certain family to a job in a different family, then a setup (whose duration depends on both family types) may be incurred. Rolling operations in steel making,[2] machine re-tooling,[3] production of plastics with colour groups,[4] aircraft landing sequencing,[5] production scheduling,[6] and PCB manufacturing[7] are some of the practical applications where sequence-dependent setup times are observed.

We specifically consider the problem where n jobs are grouped into mutually exclusive and collectively exhaustive sets. Each set of jobs is referred to as a *family*. $\mathcal{J}$ denotes the set of jobs and $\mathcal{J}(k)$ denotes the set of jobs whose family is k, $k=1,2,\ldots,K$, where K denotes the number of families. There are n_k jobs in $\mathcal{J}(k)$, hence $\sum_{k=1}^{k} n_k = n$. Job j has a processing time p_j, $j=1,2,\ldots,n$, and all jobs are available at time zero. If the sequence requires a switch from a job in Family k to a job in Family l, then a setup time of $s_{k,l}$ units is incurred, where $s_{k,l}$ need not be equal to $s_{l,k}$. By definition $s_{k,k}$ equals 0, $k=1,2,\ldots,K$, and $s_{0,k}$, $k=1,2,\ldots,K$, denotes the setup time required when a job that belongs to Family k is sequenced in the first position in a sequence. The objective is the minimization of the sum of the completion times of the jobs, or $\sum_{j=1}^{n} C_j$, where C_j is the completion time Job j. Based on the classification scheme of Lawler *et al*,[8] the problem we address in this paper is denoted by $1/s_{ik}/\sum_j C_j$. Rinnooy Kan[9] has shown that the Directed Hamiltonian Path problem, which is NP-complete, can be reduced to a special case of the $1/s_{ik}/\sum_j C_j$ problem. Ahn and Hyun[2] present a $O(K^2 n^K)$ dynamic programming solution procedure for the $1/s_{ik}/\sum_j C_j$ problem. From a theoretical viewpoint, Ahn and Hyun's[2] solution procedure indicates that the problem is solvable in polynomial time for a fixed value of K. However, as stated in Ahn and Hyun,[2] the implementation of the dynamic program becomes intractable even with a moderate number of families. Ahn and Hyun,[2] and Gupta[10] present heuristic solution procedures for the $1/s_{ik}/\sum_j C_j$ problem.

Monma and Potts[11] present a dynamic programming algorithm for the $1/s_{ik}/\sum_j w_j C_j$ problem with a time complexity of $O(K^2 n^{K^2+K})$. Ghosh[12] studies the same problem and develops an optimal solution algorithm with a time complexity of $O(K^2 n^{2K})$. Mason and Anderson,[3] Crauwels *et al*,[13] and Dunstall *et al*[14] present B&B algorithms for the closely related $1/s_i/\sum_j w_j C_j$ problem, where setup times are not sequence dependent. For the same problem, Mason,[15] Crauwels *et al*[13,16] develop heuristic solution procedures.

**Correspondence: S Karabatı, College of Administrative Sciences and Economics, Koç University, Rumelifeneri Yolu, Sarıyer, İstanbul 34450, Turkey.*
E-mail: skarabati@ku.edu.tr

We present a new lower bounding scheme which is based on a network formulation of the problem in the next section. The lower bound is obtained by solving an Integer Programming (IP) formulation of the network problem via Lagrange relaxation. The size of the IP formulation depends on the size of the network representation of the scheduling problem, and we discuss a set of rules that could drastically reduce the size of the network. Following that section, we also present a procedure to obtain upper bounds on the optimal value of the scheduling problem. The upper bounding procedure uses the solution of the Lagrange relaxation of the IP formulation to generate feasible sequences, that is, valid upper bounds.

The B&B algorithm we present in the subsequent section is a standard implicit enumeration scheme. The search scheme for the optimal solution is guided by lower and upper bounds generated through solution of the Lagrange relaxation of the IP formulation. In the last two sections, results of the computational experiments with the new procedure are reported, and concluding remarks are presented.

Lower bound computation

Our lower-bound computations are based on a minimum-cost network flow formulation of the problem. Solving the linear programming relaxation of this formulation would give us a tight lower bound at the root node of the B&B tree. However, as discussed later in this section, instead of directly solving the LP formulation of the problem using the simplex method, we present a Lagrangean relaxation based approximate solution procedure to obtain lower and upper bounds on the optimal solution value much more efficiently.

We define a transshipment type network, $G(V, A)$, where there is a dummy source/sink node, and there are n *'levels'* of nodes, one for each position in a sequence. More specifically, we define $V=\bigcup_{p=0}^{n} V_p$, where V_p is the set of nodes at level p. By definition, $V_0=\{0\}$ is the dummy source/sink node. Furthermore, $A=\bigcup_{p=0}^{n} A_p$, where A_p is the set of arcs that originate from nodes in V_p and go into nodes in V_{p+1}, for $p<n$ (for $p=n$, all arcs go into V_0).

A node $v \in V_p$ corresponds to a job, denoted by $J(v)$, being scheduled in position p. Each arc that originates from v corresponds to a job being the immediate successor of $J(v)$ in a sequence. Hence at most n arcs could originate from any node. Furthermore, all arcs in A_p that correspond to Job j go into the same Node w with $J(w)=j$. Therefore, a path from source to sink corresponds to a feasible sequence of n jobs when each job appears exactly once on this path. The family of Job j is denoted by $F(j)$ and family of the job of Node v is denoted by $F(v)$, rather than $F(J(v))$, in order to simplify the notation. The cost of an arc that goes from Node $v \in V_{(p-1)}$ into a Node $w \in V_p$ is equal to $c_{v,w}=(n-p+l)*(s_{k,r}+p_{J(w)})$, where $k=F(v)$ and $l=F(w)$. Note that an arc that goes from Node $v \in V_{(p-1)}$ into a Node $w \in V_p$ corresponds to assigning $J(w)$ into the pth position of the sequence, after $J(v)$ has been assigned into the p–1st position, with a total processing time of $s_{k,l}+p_{J(w)}$. Since the processing time of job that is assigned to the pth position affects the completion times of jobs that are in positions $p, p+1, \ldots, n$, its overall contribution to the objective function would be $(n-p+1)*(s_{k,l}+p_{J(w)})$.

We illustrate the network representation of the problem with a 3-job, 2-family (ie, $n=3$ and $K=2$) problem with the following data: $\mathscr{J}(1)=\{1,2\}$, $\mathscr{J}(2)=\{3\}$ (ie, Jobs 1 and 2 belong to first family, and Job 3 belongs to the second family); $s_{0,1}=s_{0,2}=s_{1,1}=s_{2,2}=0$, $s_{1,2}=3$, $s_{2,1}=4$; $p_1=3$, $p_2=4$, $p_3=2$. Figure 1 illustrates the network representation of the problem. The nodes of the network are clustered in *levels* as $V_1=\{1,2,3\}$, $V_2=\{4,5,6\}$, $V_3=\{7,8,9\}$. Each level represents sequencing decisions for a certain position in the sequence, that is, the nodes in V_i correspond to possible assignments into the ith position in the sequence. For example, Node 2 denotes that Job 2 is assigned to the first position in the sequence, and Node 9 denotes that Job 3 is assigned to third position in this 3-job problem. A path between the source and sink nodes corresponds to a sequence if nodes of Job i, $i=1,2,\ldots,n$, are visited only once. The path 0–1–5–9 corresponds to the feasible sequence of Job 1-Job 2-Job 3. The path 0–1–5–7, on the other hand, is not feasible because Job 1 is assigned to both the first and last positions. With the sequence Job 1-Job 2-Job 3, the completion times of Jobs 1, 2, and 3 are $C_1=s_{0,1}+p_1=0+3=3$, $C_2=s_{0,1}+p_1+s_{1,1}+p_2=0+3+0+4=7$, and $C_3=s_{0,1}+p_1+s_{1,1}+p_2+s_{1,2}+p_3=0+3+0+4+3+2=12$, respectively. Therefore, the sum of completion times is equal to $C_1+C_2+C_3=3(s_{0,1}+p_1)+2(s_{1,1}+p_2)+(s_{1,2}+p_3)$. We note that assigning Job i to the first position affects the completion times of all jobs that are assigned after itself, along with its own completion time. Therefore the cost of an arc between Nodes 0 and v in the first level of the network would be equal to $3(s_{0,F(v)}+p_{J(v)})$. In general, the cost of an arc between Nodes w and v, where $w \in V_{p-1}$, and $v \in V_p$, would be $(n-p+1)(s_{F(w),F(v)}+p_{J(v)})$.

In this network representation of the problem, the maximum number of nodes and arcs of the network would

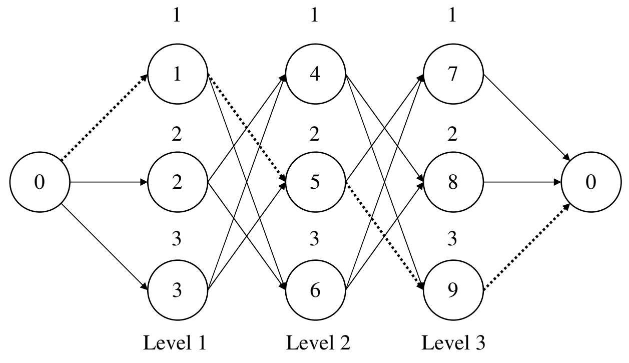

Figure 1 Network Representation of the First Example.

be $O(n^2)$ and $O(n^3)$, respectively. The optimization problem we need to solve on the network is a shortest path problem with the additional *feasible sequence* constraint (the nodes of Job i, $i = 1, 2, \ldots, n$, should be visited only once), and it would be difficult to solve large problems with this initial network. In the subsequent sections we will first formally state the optimization problem, and then discuss rules that will be employed to reduce the number of nodes and arcs in the network

The integer programming formulation

Given this network structure, we can formulate the problem as an integer program, as follows: Let $V = \{0, 1, \ldots, m\}$ denote the set of nodes, where 0 is the dummy source/sink node, and $A = \{(v, w): v, w \in V\}$ denote the set of arcs. For a given Node v let $P(v) \in \{0, 1, \ldots, m\}$ denote the level of the node in the graph. For a given arc (v, w), defining $X_{v,w} \in \{0,1\}$ to be the flow on the arc, and $c_{v,w}$ to be its cost (where, by definition $c_{v,0} = 0$ for all $v \in V_n$), we can write the following formulation:

$$\begin{aligned} \min \quad & \sum_v \sum_w c_{v,w} X_{v,w} \\ \text{s.t.} \quad & \end{aligned} \tag{1}$$

$$\sum_v X_{v,w} = \sum_v X_{w,v}, \quad w \in V \tag{2}$$

$$\sum_v \sum_{w:J(w)=j} X_{v,w} \leqslant 1, \quad j = 1, 2, \ldots, n \tag{3}$$

$$\sum_v X_{0,v} = 1, \tag{4}$$

$$X_{v,w} \in \{0, 1\}, \quad v, w \in V \tag{5}$$

Constraints (2) are the flow conservation constraints at each node. Constraints (3) require that no more than one arc corresponding to each job have a unit-flow. Finally, Constraint (4) requires a unit-flow out of the source/sink node. Note that, Constraints (3) can be stated as inequalities, because, due to structure of the network and Constraint (4), unit-flow will occur on one arc at each level of the network, and, since no two arcs of the same job can have unit flow, exactly one arc for each job will have a unit-flow. Clearly, this is a transshipment model with additional side constraints given in (3), and solving the linear programming relaxation of this formulation would give a lower bound to our scheduling problem.

Reducing the size of the network

The actual number of nodes and arcs can be drastically reduced with the intra-family SPT property, which states that in an optimal schedule Job i in Family k precedes Job j of the same family if $p_i < p_j$ (see Ahn and Hyun[2]). In this section, we discuss how the intra-family SPT and other properties of the problem can be used to reduce the size of the network representation of the problem.

We will use the following problem instance to demonstrate the properties we use in reducing the size of the network: $n = 7$, $f = 3$, $\mathscr{J}(1) = \{1, 2, 3, 4, 5\}$, $\mathscr{J}(2) = \{6\}$, $\mathscr{J}(3) = \{7\}$. p_j equals 1, 3, 2, 3, 5, 3, 3 for Jobs 1 through 7, respectively. The setup times are $s_{1,2} = 1$, $s_{1,3} = 5$, $s_{2,1} = 5$, $s_{2,3} = 3$, $s_{3,1} = 4$, $s_{3,2} = 2$. The network depicted in Figure 2 corresponds to this instance.

The first method of reducing the size of the network is reducing the number of arcs created out of Node v by identifying the set of jobs that are 'definitely' sequenced up to Node v (including the sequencing decision that corresponds to Node v), which is denoted by $\mathscr{D}(v)$. By definition, if $j \in \mathscr{D}(v)$ then Job j appears in all partial sequences ending at Node v. However, it is important to note that there can exist a job k that is in all partial sequences ending at Node v but not in $\mathscr{D}(v)$. We let $\mathscr{D}(v,k) = \{j: j \in \mathscr{D}(v) \text{ and } F(j) = k\}$. Due to the definition of $\mathscr{D}(v)$, no arc corresponding to jobs in $\mathscr{D}(v)$ would be created out of v.

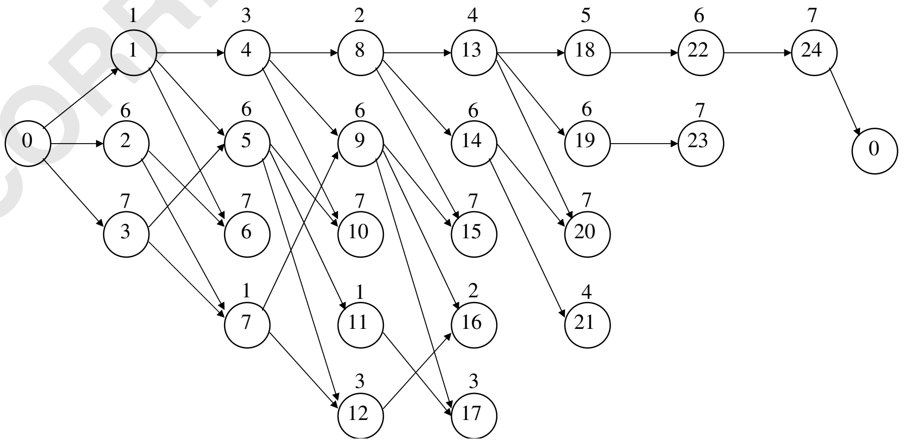

Figure 2 Network Representation of the Second Example.

One of the sets used in determining $\mathscr{D}(v)$ is $\mathscr{A}(v)$, the set of jobs on all source-to-v paths in the network. Assuming $w \in V_{(p+1)}$,

$$\mathscr{A}(w) = \left\{ J(w), \bigcup_{v \in V_p : (v,w) \in A_p} \mathscr{A}(v) \right\}, \quad \text{where } \mathscr{A}(0) = \emptyset$$

We let $\mathscr{A}(v, k) = \{j : j \in \mathscr{A}(v) \text{ and } F(j) = k\}$.

In the example network, $\mathscr{A}(12) = \{1, 3, 6, 7\}$ since $\mathscr{A}(5) = \{1, 6, 7\}$, $\mathscr{A}(7) = \{1, 6, 7\}$ and $J(12) = 3$. Therefore, $\mathscr{A}(12, 1) = \{1, 3\}$, $\mathscr{A}(12, 2) = \{6\}$ and $\mathscr{A}(12, 3) = \{7\}$.

Given $\mathscr{A}(v)$, where $v \in V_p$, we determine $\mathscr{D}(v, k)$ as follows: First, by assuming all jobs in $\mathscr{A}(v)$ that belong to families other than k are scheduled up to Node v, we determine the maximum number of Family k jobs that can be in $\mathscr{D}(v, k)$, as $m(v, 242111k) = \min(|\mathscr{A}(v)|, p) - \bar{f}(v, k)$, where $\bar{f}(v, k) = |\cup_{l : l \neq k} \mathscr{A}(v, l)|$. Owing to Domination Rule 3 discussed below, $|\mathscr{A}(v)| \geqslant p$, thus $m(u, k) = p - \bar{f}(v, k)$. Then, we create the set $\mathscr{M}(v, k)$ as follows. If $F(v) = k$ the set contains $m-1$ shortest processing time jobs in $\mathscr{A}(v, k)$ excluding $\mathscr{J}(v)$, otherwise it includes—shortest processing time jobs $\mathscr{A}(v, k)$.

In Example Network 1, consider Node $20 \in V_5$, for which $\mathscr{A}(20) = \{1, 2, 3, 4, 6, 7\}$ and $\mathscr{A}(20, 1) = \{1, 2, 3, 4\}$. Since $\bar{f}/(20, 1) = 2$, there are $5-2 = 3$ positions available for Family 1 jobs in the partial sequence up to Node 20. Since $J(20) = 7$, whose family is 3, $\mathscr{M}(20, 1) = \{1, 3, 2\}$. Note that Job 2 or 4 could have been in the set since both have the same processing time of 3, but one is selected arbitrarily. However, we cannot be sure that Job 2 is going to be scheduled up to Node 20, as discussed below.

Now let us assume the maximum processing time of jobs in $\mathscr{M}(v, k)$ is $p_{\max}$. As a result of the intra-family SPT rule, we know that job(s) of duration $p_{\max}$ will be the last jobs of Family k in the partial sequence up to Node v. However, if there exists a job, say Job j' of Family k not in $\mathscr{M}(v, k)$ but in $\mathscr{A}(v, l) \setminus J(v)$, whose processing time is also $p_{\max}$, we cannot know definitely whether the jobs with duration $p_{\max}$ that are in $\mathscr{M}(v, k)$ or Job j' will actually be scheduled, so we remove all of them from $\mathscr{M}(v, k)$. Then, we set $\mathscr{D}(v, k) = \mathscr{M}(v, k) \bigcup J(v)$, if $F(v) = k$ or $\mathscr{D}(v, k) = \mathscr{M}(v, k)$, otherwise.

Continuing the previous example for Node 20, since Job 4 is in $\mathscr{A}(20, 1) \setminus \{7\}$ but not in $\mathscr{M}(20, 1)$, either Job 4 or 2 could be the last job in the partial sequence up to Node 20. Therefore, we cannot say definitely which one of them will be scheduled and we update $\mathscr{M}(20, 1)$ to be $\{1, 3\}$. For Family 2, since $\bar{f}(20, 2) = 5$, it is possible that no jobs of this family are scheduled up to Node 20, making $\mathscr{M}(20, 2) = \emptyset$ and $\mathscr{D}(20, 2) = \emptyset$. Finally, $\mathscr{M}(20, 3) = \emptyset$ as well, since $\bar{f}(20, 2123) = 5$. However, $\mathscr{D}(20,3) = \{7\}$ as $J(20) = 7$. Hence, $\mathscr{D}(20) = \{1, 3, 7\}$.

It is important to note that the set $\mathscr{D}(20)$ is actually a subset of the actual number of definitely scheduled jobs. Looking at the network in Figure 2 we can clearly see that there are two partial sequences leading to Node 20 and which ever is selected, Jobs 1, 2, 3 and 7 would be scheduled. However, since we determine $\mathscr{D}(20)$ using sets not partial sequences, we can only identify $\{1, 3, 7\}$.

Having determined the set $\mathscr{D}(v)$, the next step is to determine the set of candidate jobs, $\mathscr{C}(v)$, for which we could create arcs out of Node v. $\mathscr{C}(v) = \bigcup_{k \in \{1 \ldots K\}, t} \mathscr{C}(v, k, t)$, where $\mathscr{C}(v, k, t)$ is the set of jobs of Family k with processing time t that are *candidates* for arcs out of Node v.

Clearly, we would consider creating arcs only for jobs that are not in $\mathscr{D}(v)$. However, a further reduction in the number of arcs is possible due to the following observation. Let $\mathscr{J}(k, t)$ be the set of jobs of Family k with processing time t. Given a sequence, if we interchange the positions of two jobs $i, j \in \mathscr{J}(k, t)$ the sum of completion times would remain the same. So, when we select the jobs of Family k, for which we are going to create arcs out of Node v, we do the following: We determine $\mathscr{F}(v, k, t) = \mathscr{J}(k, t) \setminus \mathscr{D}(v, k)$, the set of jobs of Family k that have the same processing time t and could be assigned to an arc out of Node v. Since we would like to reduce the size of the network as much as possible we would like to create an arc corresponding to only one of these jobs. This could only be possible if there is at least one Job j such that $j \in \mathscr{F}(v, k, t)$ and $j \notin \mathscr{A}(v, k)$. If there is such a Job j then we set $\mathscr{C}(v, k, t) = \{j\}$, otherwise we let $\mathscr{C}(v, k, t) = \mathscr{F}(v, k, t)$.

We first consider the source node in our example. Recall that $\mathscr{J}(1, 3) = \{2, 4\}$. Since $\mathscr{D}(v, k) = \emptyset$, $\mathscr{F}(0, 1, 3) = \{2, 4\} \setminus \emptyset = \{2, 4\}$. This means we can have an arc for both Jobs 2 and 4 out of the source node. However, since $\mathscr{A}(0, k) = \emptyset$, creating an arc for one of the two would not eliminate any optimal solution as the other could be scheduled later. Hence, $\mathscr{C}(0, 1, 3) = \{2\}$. Since for the other jobs there are no such alternatives $\mathscr{C}(0) = \{1, 2, 3, 5, 6, 7\}$. Since source node is a special case, we also consider Node 9. Recall that $J(9) = 6$ and $F(9) = 2$. $\mathscr{F}(9, 1, 3) = \mathscr{J}(1, 3) \setminus \mathscr{D}(9, 1) = \{2, 4\} \setminus \{1, 6\} = \{2, 4\}$. Since $\mathscr{A}(9, 1) = \{1, 3, 6, 7\}$, $\mathscr{C}(9, 1, 3) = \{2\}$, leading to $\mathscr{C}(9) = \{2, 3, 5, 7\}$.

After $\mathscr{C}(v)$ is determined, arcs for some of these jobs may still be dominated due to the following rules:

Rule 1 For Node $v \in V_p$, this rule calculates two values for each family, namely, $S_{\max}(v, k)$, a lower bound on the maximum processing time of jobs of Family k that would be scheduled up to and including Node v and $U_{\min}(v, k)$, an upper bound on the minimum processing time of jobs of Family k that would be scheduled after Node v. An arc for Job j with $F(j) = k$ out of Node v is dominated if $p_j < S_{\max}(v, k)$ or $p_j > U_{\min}(v, k)$ due to the intra-family SPT rule.

$S_{\max}(v, k)$ equals the largest duration of jobs in $\mathscr{D}(v, k)$.

In calculating $U_{\min}(v, k)$ we assume jobs in $\mathscr{A}(v, k)$ are scheduled before any other job in $\mathscr{A}(v)$. Thus, we sort the jobs in $\mathscr{A}(v, k) \setminus J(v)$, if $F(v) = k$ or just $\mathscr{A}(v, k)$, otherwise, in increasing processing time. Denoting this sorted set by $\mathscr{S}(v, k)$, if $|\mathscr{S}(v, k)| \geqslant p$, that means some of the jobs in $\mathscr{S}(v, k)$ will be left unscheduled by Node v. Then, $p_{v,k,[p]}$, denoting the pth largest processing time job in $\mathscr{S}(v, k)$ would be the minimum processing time of jobs in $\mathscr{A}(v,k)$ left unscheduled. Jobs in $\bar{\mathscr{A}}(v,k) = \mathscr{J}(k)\backslash\mathscr{A}(v,k)$ are clearly not scheduled by Node v. We let $p'24031_{v,k,[1]} = \min\{p_j : j \in \bar{\mathscr{A}}(v,k)\}$. Then, if $|\mathscr{S}(v,k)| \geqslant p$, $U_{\min}(v,k) = \min\{p_{v,k,[p]}, p'_{v,k,[1]}\}$, otherwise, $U_{\min}(v,k) = p'_{v,k,[1]}$.

For arcs out of the source node, since no jobs are scheduled yet, $U_{\min}(0,k) = \min\{p_j: j \in \mathscr{J}(k)\}$ and $S_{\max}(0,k) = 0$.

Going back to our example, for Node 0, $U_{\min}(0,1) = 1$ and arcs for Jobs 2, 3, 5 are dominated since their processing times are larger than 1. For Node 9, $\mathscr{S}(9,1) = \{1,3\}$ and $\bar{\mathscr{A}}(9,1) = \{2,4,5\}$. Since the position of Node 9 is 3 and $|\mathscr{S}(9,1)| = 2$, $U_{\min}(9,1) = p'_{9,1,[1]} = 3$. On the other hand, since $F(9) = 2$, $\mathscr{D}(9,1) = \{1\}$, $\mathscr{D}(9,2) = \{6\}$ and $\mathscr{D}(9,3) = \emptyset$, we have $S_{\max}(9,1) = 1$, $S_{\max}(9,2) = 3$, $S_{\max}(9,3) = 0$.

Rule 2 In creating a network we keep track of the length of the shortest path to each node. When creating an arc out of Node v into Node w with cost $c_{v,w}$, if the sum of the length of the shortest path to Node v and $c_{v,w}$ exceeds the upper bound on the sum of completion times, then we do not create arc (v,w).

Rule 3 When creating an arc out of $v \in V_p$ for Job j, if $|\mathscr{A}(v) \bigcup \{j\}| < p+1$, then the arc is dominated since there must be at least $(p+1)$ jobs that could be scheduled up to and including level $p+1$ of the network.

Rule 4 Let $L_j = |\{k: p_k > p_j, k \in \mathscr{J}(F(j))\}|$ and $S_j = |\{k: p_k < p_j, k \in \mathscr{J}(F(j))\}|$. When creating an arc for Job j from a node in position p, the arc is dominated if $L_j > n-p-1$ or if $S_j > p$, due to the intra-family SPT rule.

Lagrangean heuristic

The linear programming relaxation of this network formulation gives a tight initial lower bound and can be solved in a reasonable time for quite large instances. However, the structure of the formulation can be exploited to obtain these bounds much faster by Lagrangean relaxation. Specifically, when we relax Constraint Set (3), we obtain a shortest path formulation, and, if optimal Lagrange multipliers are used, the lower bound obtained by this relaxation would be equal to the one obtained by the optimum solution to the linear programming relaxation.[17] When we let $\lambda = \{\lambda_j, j = 1, 2, \ldots, n\}$ be the Lagrange multiplier vector for Constraint Set (3), the Lagrangean relaxation formulation can be written as

$$Z_D(\lambda) = \min \sum_v \sum_w (\mathscr{C}_{v,w} + \lambda_j) X_{v,w} - \sum_{j=1}^{n} \lambda_j \qquad (6)$$

s.t. (2), (4) and (5).

It is well known that quite good but not necessarily optimal Lagrangean multipliers may be obtained by the subgradient method. In the subgradient method, given λ^k as the vector of the Lagrangean multipliers at iteration k and $Z_D(\lambda^k)$ as the corresponding optimal objective function value for the Lagrangean relaxation formulation, λ^{k+1} is calculated as follows: $\lambda_j^{k+1} = \max\{0, \lambda_j^k + t_k(\sum_{v,w:J(w)=j} X_{v,w} - 1)\}$, where t_k is the scalar step size and calculated as

$$t_k = \frac{a_k(Z^* - Z_D(\lambda^k))}{\sum_j \left(1 - \sum_{v,w:J(w)=j} X_{v,w}\right)^2}$$

where Z^* is the current best upper bound and a_k is a scalar. In our implementation, Z^* is initially set to be the upper bound found by the neighbourhood search heuristic of Ahn and Hyun.[2] If the solution to the Lagrangean relaxation is feasible for the integer programming formulation (ie, one arc for each job is selected) than the subgradient algorithm stops (giving the optimal solution). Otherwise, the number of iterations is limited to $30n$. This has empirically proved to be sufficient for the Lagrangean lower bound to converge to the LP lower bound. The scalar a_k is initially set to be 2, as suggested by Fisher.[17] Then, a_{k+1} is reduced to $a_k/2$ if either the lower bound has not been improved for n iterations, or a_k has not been changed for $3n$ iterations. Initially, the Lagrange multipliers are set to 0.

We use the following heuristic to find an upper bound in each iteration of the Lagrangean lower bounding procedure. When the problem is solved with a set of Lagrange multipliers, a shortest path from the source node to the sink node is generated. Only one arc is chosen in every level of the network, therefore one assignment is made to every position in the sequence. However, because Constraint Set (3) is relaxed, these assignment may not be feasible, that is, one job may be assigned to more than one position. Given this possibly infeasible sequence, we can then determine 'family-to-position' assignments, that is, find the family of job that is assigned to position j, $j = 1, 2, \ldots, n$, in the sequence, and build a feasible sequence around these assignments. The heuristic is based on the observation that, if we use the family-to-position assignments from the Lagrangean lower bound to obtain a feasible set of family-to-position assignments, that is, one in which Family k is assigned to exactly n_k positions, then due to the intra-family SPT rule we can easily create a sequence with minimum cost for the given family-to-position assignments. Specifically, the heuristic is designed as follows.

Since a lower bound found at an iteration of the Lagrangean heuristic is a shortest path through the network, the shortest path corresponds to a single arc selected at each position. We let $L(p)$ denote the job of the arc that goes into a node $w \in V_p$ that appears in the shortest path. From this information we can identify $\mathscr{P}(k) = \{p: F(L(p)) = k, p = 1, \ldots, n\}$, the set of positions for which a job that belongs to Family k has been selected in the lower bound. If $|\mathscr{P}(k)| > n_k$, then we have positions in excess of the requirement and $\mathscr{E}$ is the set of such families with an excess number of positions. On the other hand, $\mathscr{D}$ is the set of families for which $|\mathscr{P}(k)| \leqslant n_k$.

For each family $k \in \mathscr{E}$, the heuristic assigns jobs of that family in order of increasing processing time to the positions (in increasing order of position index) and stores left-over positions of the family into a set $\mathscr{F}$. After this is completed for all families in $\mathscr{E}$, the set $\mathscr{F}$ contains all newly available positions for the families in $\mathscr{D}$. In the next step, the number of positions that each family $k \in \mathscr{D}$ needs is calculated as $d(k) = n_k - |\mathscr{P}(k)|$. Let $f_{[i]}$ be the ith lowest indexed family in $\mathscr{D}$. Then, the first $d(f_{[1]})$ positions in $\mathscr{F}$ are inserted into $\mathscr{P}(f_{[1]})$, the next $d(f_{[2]})$ positions are inserted into $\mathscr{P}(f_{[2]})$ and so on, until all positions in $\mathscr{F}$ are allocated to some family in the set of families in $\mathscr{D}$. Since now each family $k \in \mathscr{D}$ has n_k positions in its set $\mathscr{P}(k)$, we can assign its jobs to these positions in SPT order.

We will demonstrate the Lagrangean heuristic using the example network provided in Figure 3. The data for this problem instance are as follows: $n = 7$, $f = 3$, $\mathscr{J}(1) = \{1, 2, 3, 4\}$, $\mathscr{J}(2) = \{5\}$, $\mathscr{J}(3) = \{6, 7\}$. p_j equals 1, 2, 1, 3, 3, 2, 5 for jobs 1 through 7, respectively. The setup times are $s_{1,2} = 4$, $s_{1,3} = 5$, $s_{2,1} = 3$, $s_{2,3} = 3$, $s_{3,1} = 3$, $s_{3,2} = 4$.

The initial upper bound for this problem is found to be 71 with the sequence 1, 3, 2, 4, 5, 6 and 7. Using Lagrangean multipliers that are equal to 0, the first lower bound for the network corresponds to the path through the nodes (jobs) 0, 3 (6), 7 (1), 12 (3), 19 (1), 23 (2), 27 (4), 30 (7), 0 with arc costs 14, 24, 5, 4, 6, 6, 10 and 0, respectively. The length of this shortest path is 69. Since all the Lagrangean multipliers are zero, 69 is the lower bound as well. Since Job 1 is assigned to two positions and Job 5 is not assigned to any position, this path does not yield a feasible solution. The upper bound heuristic picks up the job sequence of this shortest path to find a feasible sequence as follows: Positions 2, 3, 4, 5 and 6 have job of Family 1, thus $\mathscr{P}(1) = \{2, 3, 4, 5, 6\}$, whereas, $\mathscr{P}(2) = \emptyset$ and $\mathscr{P}(3) = \{1, 7\}$. Since $n_2 = 1 > |\mathscr{P}(2)|$, $\mathscr{D} = \{2\}$. On the other hand, since $n_1 = 4 \leqslant |\mathscr{P}(1)|$ and $n_3 = 2 \leqslant |\mathscr{P}(3)|$, we have $\mathscr{E} = \{1, 3\}$. Hence, first we assign Jobs 1, 3, 2, 4 of Family 1 (in SPT order) to positions 2, 3, 4, 5, respectively, and insert position 6 into $\mathscr{F}$. Then we assign Jobs 6, 7 of Family 3 to positions 1, 7, respectively. Having assigned jobs of all the families in $\mathscr{E}$, the set $\mathscr{F} = \{6\}$ now contains the only newly available position for the Family 2. Since Family 2 is the only family with deficient number of positions $|\mathscr{F}| = d(2) = 1$. So we insert the position in $\mathscr{F}$ into $\mathscr{P}(2)$ to obtain $\mathscr{P}(2) = \{6\}$. Then we assign the only job of Family 2 to this position, leading to the following complete sequence for the problem: 6, 1, 3, 2, 4, 5, 7. The sum of completion times for this sequence is 82, so still the best upper bound is 71. The second iteration of the Lagrangean heuristic starts with the update of the Lagrangean multipliers as follows:

$$t_k = \frac{2(71 - 69)}{((1-2)^2 + (1-1)^2 + (1-1)^2 + (1-1)^2 + (1-0)^2 + (1-1)^2} = 2$$

leading to $\lambda_1^1 = \max\{0, 0 + 2(2-1)\} = 2$, $\lambda_2^1 = \max\{0, 0 + 2(1-1)\} = 0$, $\lambda_3^1 = \max\{0, 0 + 2(1-1)\} = 0$, $\lambda_4^1 = \max\{0, 0 + 2(1-1)\} = 0$, $\lambda_5^1 = \max\{0, 0 + 2(0-1)\} = 0$, $\lambda_6^1 = \max\{0, 0 + 2(1-1)\} = 0$, $\lambda_7^1 = \max\{0, 0 + 2(1-1)\} = 0$. With these new Lagrangean multipliers, only the costs of the arcs of Job 1 are increased by 2. The shortest path with these costs goes through the nodes (jobs) 0, 2 (1), 4 (3), 9 (2), 14 (4), 21 (5), 26 (6), 30 (7), 0, with a length of 73. Since sum of the Lagrangean multipliers is 2, the lower bound is $73 - 2 = 71$. Note that the sequence of jobs corresponding to this path

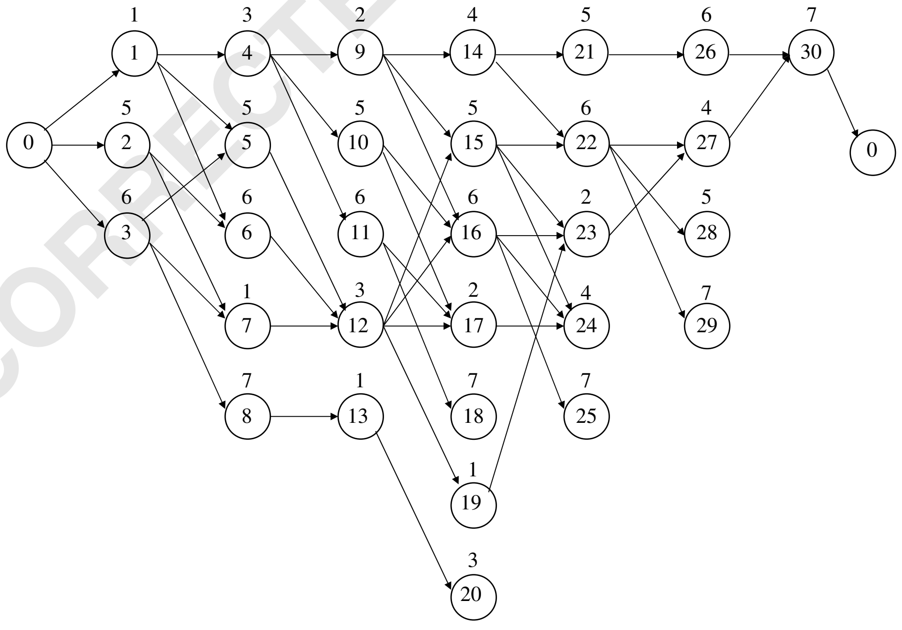
Figure 3 Network Representation of the Third Example.

assigns each job to a unique position, hence it is feasible with a sum of completion times of 71. Thus, this sequence is optimal.

The branch-and-bound algorithm

In this section we present a branch-and-bound (B&B) algorithm to exactly solve the problem. The B&B algorithm performs its search over the network of the problem developed in the lower bound computation section. A node in the B&B tree corresponds to a partial sequence of l jobs with $\sigma_l = \{j_1, \ldots, j_l\}$, $1 \leqslant l \leqslant n$, or a path from the dummy source/sink node to a certain node of the network at level l where no two arcs that are on the path belong to the same job. A lower bound at a node of the B&B tree can be easily computed by adding the sum of completion times value of the partial sequence to the objective function value (Equation (6)) of the Lagrangean relaxation formulation starting from the last node of the path that corresponds to the partial sequence. Note that, the length of the shortest path from the last node of the partial sequence to the dummy source/sink node minus the sum of the Lagrange multipliers of the jobs that are not in the partial sequence would give the desired objective function value.

The B&B algorithm imports the network structure and shortest path values of the nodes from the initial lower bound computation stage, and starts with an initial list of single-job sequences that correspond to nodes in the first level of the network. The partial sequences in the initial list are sorted in decreasing order of lower bound values. The branching scheme removes the first partial sequence from the list and expands it following the arcs that emanate from the node that corresponds to the last job in the partial sequence. An arc can be a candidate for expanding the partial sequence only if the job that corresponds to the end node of the arc is not in the partial sequence. If the lower bound of a newly created partial sequence is smaller than the current upper bound, the partial sequence is inserted to the list of partial sequences. The insertion process is designed as follows: until the one-millionth partial sequence is generated, the first $n \times 100$ partial sequences are sorted in decreasing order of lower bound values, and if a partial sequence's lower bound value is greater that of the partial sequence in the $(n \times 100)$th position in the list, it is inserted in position $n \times 100 + 1$. After the one-millionth partial sequence is generated, only the first 10 partial sequences are sorted in decreasing of lower bound values, and if the lower bound value of a newly generated partial sequence is greater than that of the 10th partial sequence in the list, it is inserted in the 11th position.

The B&B algorithm uses complete sequences that are generated during the search to update the upper bound value. The initial upper bound of the B&B algorithm is the sum of completion times value of the sequence generated by the Lagrangean Heuristic. At other nodes of the B&B tree, complete sequences are generated by using a modified version of the Lagrangean heuristic described earlier. The modified version of the heuristic accounts for the jobs that are already assigned to a position in the partial sequence.

In order to further improve the upper bound quality, we employ a block-insertion heuristic. This heuristic is used at the root node of the B&B tree, and when a new complete sequence that improves the current upper bound is generated during the search. The block-insertion heuristic can be outlined as follows: A complete sequence can be viewed as collection of blocks where each block refers to a group of jobs that are contiguously sequenced and belong to the same family. The block-insertion heuristic iteratively divides a block of k jobs into two sub-blocks: first l jobs and remaining $k-l$ jobs where $l = 1, 2, \ldots, k-1$. Let p be the position of the first job of the block in the original sequence. The first block of l jobs is then removed from the sequence and then re-inserted to the sequence starting from position i, $i = 1, 2, \ldots, p-1$, or from position i, $i = p+k+1,\ p+k+2, \ldots, n$, *in* the sequence. If $i<p$, the jobs that are originally sequenced in positions i, $i+1, \ldots, p-1$, are right-shifted by l positions, and if $i>p+k+1$, the jobs that are originally sequenced in positions $p+l, p+l+1, \ldots, i$, are left-shifted by l positions to make up room for the block that is going to be inserted. After the block is inserted in its new position, the sequence is viewed as blocks of contiguous positions that belong to different families, and the jobs of each family are re-assigned to positions that belong to that family using the intra-family SPT rule. The worst-case time complexity of a block-insertion operation is equivalent to determining the sum of completion times value of n sequences, and the maximum number of sub-blocks in a given sequence is equal to n. The sum of completion times value of a sequence can be determined in $O(n)$ time, therefore, for a given initial sequence, the time complexity of the heuristic is $O(n^3)$. In the current implementation of the block-insertion heuristic, if the sum of completion times value of the initial sequence is improved with a block-insertion operation, the heuristic is re-started with the newly formed sequence.

Computational results

In this section, we present a computational analysis of the B&B algorithm's performance on randomly generated problem instances. The problem instances are generated using three values of number of families ($K = 8, 12$ and 16), two values of number of jobs ($n = 50$ and 60), two values for the relative sizes of the families ($\max_{1 \leqslant k \leqslant K} n_k / \min_{1 \leqslant k \leqslant K} n_k \approx 1$ that is, approximately equal number of jobs per family, and $\max_{1 \leqslant k \leqslant K} n_k / \min_{1 \leqslant k \leqslant K} n_k \in [2, 3]$),

Table 1 Computational results: 50 job problems

												CPU Times		
				LB quality		*UB quality*		*Network size*			*Network*	*B&B*	*Total*	
K	$\frac{max_k n_k}{min_k n_k}$	p_j	$s_{l,k}$	*Average (%)*	*Maximum (%)*	*Average (%)*	*Maximum (%)*	*Average no of Nodes*	*Average no of Arcs*	*Average no of B&B Nodes*	*Average*	*Average*	*Average*	*Maximum*
8	≈1	[1,50]	[1,50]	0.14	0.87	0.18	2.48	2240.00	9236.80	20 193.90	21.51	0.78	22.29	46.29
			[1,100]	0.51	3.40	0.58	3.75	2239.92	9250.90	289 983.68	23.24	20.51	43.75	501.49
		[1,100]	[1,50]	0.30	1.40	0.55	3.64	2238.30	9478.54	153 457.12	28.47	12.56	41.03	236.28
			[1,100]	0.25	2.06	0.29	2.12	2238.20	9490.94	80 359.46	27.56	5.54	33.10	119.94
	∈[2,3]	[1,50]	[1,50]	0.22	1.41	0.21	3.52	2221.16	9837.40	34 620.50	24.41	2.52	26.92	121.98
			[1,100]	0.47	4.31	0.70	5.03	2220.84	9838.12	430 824.82	26.07	20.09	46.16	547.71
		[1,100]	[1,50]	0.19	1.07	0.46	2.76	2219.92	10 048.92	109 434.26	30.53	8.07	38.60	233.04
			[1,100]	0.25	1.64	0.35	2.34	2220.08	10 054.94	40 930.42	23.48	1.77	25.24	75.21
12	≈1	[1,50]	[1,50]	0.45	1.32	1.03	4.01	2342.56	6379.60	293 747.02	37.80	27.38	65.17	595.20
			[1,100]	0.71	2.61	1.76	9.48	2342.44	6381.60	959 779.98	35.19	76.32	111.52	883.82
		[1,100]	[1,50]	0.18	0.90	0.65	3.77	2342.04	6433.04	68 678.02	36.92	5.52	42.44	167.98
			[1,100]	0.38	1.24	1.04	5.46	2341.88	6458.88	233 657.82	37.34	22.44	59.78	443.81
	∈[2,3]	[1,50]	[1,50]	0.40	1.90	0.93	4.61	2315.64	7237.22	423 852.46	35.25	25.40	60.64	662.15
			[1,100]	0.64	3.37	1.07	5.16	2316.38	7237.20	1 861 099.78	32.82	85.02	117.84	2791.51
		[1,100]	[1,50]	0.19	0.79	0.62	3.81	2314.64	7384.82	73 883.00	34.61	5.97	40.58	117.20
			[1,100]	0.35	1.57	0.90	5.51	2315.00	7387.42	222 814.14	35.14	18.15	53.29	446.52
16	≈1	[1,50]	[1,50]	0.38	1.42	1.33	6.14	2393.90	4665.52	1 170 000.10	40.73	71.91	112.63	1573.93
			[1,100]	0.66	2.19	2.48	11.57	2394.00	4660.34	1 853 161.38	41.17	93.70	134.87	1537.34
		[1,100]	[1,50]	0.29	1.00	1.41	4.67	2393.36	4730.00	216 217.00	40.95	25.07	66.02	608.81
			[1,100]	0.53	1.98	1.82	4.61	2393.68	4691.44	4 808 053.10	43.27	233.45	276.72	7753.02
	∈[2,3]	[1,50]	[1,50]	0.45	1.79	1.81	7.65	2384.44	5007.02	448 250.08	40.67	40.77	81.44	513.42
			[1,100]	0.58	1.99	2.30	12.31	2384.36	4999.26	884 089.10	40.20	49.06	89.26	1101.31
		[1,100]	[1,50]	0.24	0.90	1.27	4.75	2384.16	5065.02	781 002.30	40.48	48.72	89.20	1184.42
			[1,100]	0.24	0.90	1.27	4.75	2384.16	5065.02	781 002.30	40.48	50.50	90.98	1197.47

Table 2 Computational results: 60 job problems

											CPU Times			
				LB quality		*UB quality*		*Network size*			*Network*	*B&B*	*Total*	
K	$\frac{max_k n_k}{min_k n_k}$	p_j	$s_{l,k}$	*Average (%)*	*Maximum (%)*	*Average (%)*	*Maximum (%)*	*Average no of Nodes*	*Average no of Arcs*	*Average no of B&B Nodes*	*Average*	*Average*	*Average*	*Maximum*
8	≈ 1	[1,50]	[1,50]	0.26	1.66	0.32	2.83	3213.20	15 951.68	323 440.58	62.12	43.42	105.55	940.92
			[1,100]	0.39	2.54	0.31	3.16	3212.88	16 020.76	1 079 491.48	52.69	65.86	118.55	2408.98
		[1,100]	[1,50]	0.17	1.10	0.39	2.48	3211.06	16 341.36	458 547.54	62.25	38.26	100.51	1702.32
			[1,100]	0.28	1.84	0.26	1.89	3210.54	16 444.40	556 697.34	57.59	38.70	96.29	1424.65
	$\in[2,3]$	[1,50]	[1,50]	0.20	1.34	0.22	2.08	3173.12	17 308.88	121 929.82	53.16	12.31	65.47	369.73
			[1,100]	0.43	2.90	0.60	6.09	3174.22	17 294.56	747 396.28	52.01	52.09	104.10	1709.48
		[1,100]	[1,50]	0.11	1.32	0.18	1.49	3169.58	17 841.38	85 097.38	51.19	14.31	65.50	624.97
			[1,100]	0.23	2.01	0.32	3.57	3171.94	17 691.16	621 013.76	49.78	43.32	93.10	1730.73
12	≈ 1	[1,50]	[1,50]	0.40	1.42	1.27	4.86	3363.46	11 049.62	604 319.06	78.27	69.01	147.28	791.92
			[1,100]	0.49	2.77	0.92	6.12	3363.72	11 018.74	2 511 797.60	69.36	179.81	249.17	2876.39
		[1,100]	[1,50]	0.27	1.15	0.76	2.81	3362.44	11 198.82	1 678 834.80	79.04	143.69	222.73	3039.69
			[1,100]	0.33	1.19	0.83	3.89	3362.18	11 236.60	777 789.00	82.35	84.19	166.54	1081.64
	$\in[2,3]$	[1,50]	[1,50]	0.31	1.28	0.79	3.67	3333.46	12 279.12	493 149.66	74.93	60.63	135.56	1165.97
			[1,100]	0.41	2.49	1.01	6.04	3335.90	12 182.26	4 635 150.48	69.88	277.62	347.50	6685.11
		[1,100]	[1,50]	0.24	0.94	0.53	2.86	3332.28	12 540.80	374 905.50	77.89	44.12	122.01	668.41
			[1,100]	0.34	0.81	1.30	5.35	3334.00	12 450.44	332 245.58	82.74	39.52	122.25	510.80
16	≈ 1	[1,50]	[1,50]	0.34	0.94	1.46	5.60	3434.68	8 368.22	1 813 900.56	86.26	180.28	266.53	2970.86
			[1,100]	0.61	2.48	2.08	8.66	3434.92	8 324.66	4 112 097.02	90.57	355.71	446.28	4589.66
		[1,100]	[1,50]	0.33	1.35	1.61	4.10	3433.94	8 470.02	4 446 052.30	90.14	387.24	477.37	7865.28
			[1,100]	0.30	1.71	1.55	7.12	3433.84	8 486.24	2 111 891.08	89.03	160.98	250.01	5305.20
	$\in[2,3]$	[1,50]	[1,50]	0.51	2.12	2.22	7.13	3431.54	8 498.22	22 373 264.22	89.52	1 152.69	1 242.20	24 853.34
			[1,100]	0.45	1.47	1.14	6.88	3431.72	8 504.08	2 190 105.24	84.12	134.62	218.73	1331.27
		[1,100]	[1,50]	0.27	0.69	1.37	4.14	3431.30	8 563.96	687 307.74	86.85	67.87	154.72	1573.84
			[1,100]	0.44	1.65	1.45	7.39	3431.14	8 586.36	9 165 105.26	87.64	470.00	557.64	9150.58

PPL.JORS.2601989

two discrete uniform distributions for setup times ($s_{l,k} \sim \text{DU}[1, 50]$, and $s_{l,k} \sim \text{DU}[1, 100]$), and two discrete uniform distributions for processing times ($p_j \sim \text{DU}[1, 50]$, and $p_j \sim \text{DU}[1, 100]$).

The algorithms were coded in C, and the test instances were solved on a SUN/SOLARIS with a 2.4 GHz processor and 5 GB RAM.

The detailed performance of the B&B algorithm is reported in Tables 1 and 2. For each number of jobs, number of families, relative sizes of families, setup and processing times distributions combination, a set of 50 problems are solved optimally. In both tables, we report the quality of the lower and upper bounds at the initial node of the search tree. The quality of upper and lower bounds is computed as follows: Let LB, H and O be the lower bound, heuristic solution, and optimal values, respectively. Then the lower (upper) bound quality is computed as

$$\frac{O - LB}{O} \times 100 \left(\frac{H - O}{O} \times 100 \right)$$

We also report the average number of nodes and arcs of the networks that are used to generate the initial lower bound values. The network sizes reported in both tables are reasonably small for the number of jobs and families considered in the computational analysis. The average CPU time to generate the network and compute the lower bound value at the initial node is around 40 (80) s in 50 (60) job problems.

As expected, we clearly see that the most important parameters that affect the difficulty of solving these problems are the number of families and the number of jobs. Interestingly, we observe that the performance of the algorithm is not significantly affected by the remaining three parameters.

Overall, we see that the performance of the B&B algorithm is quite good, and only when the number of jobs is 60 and the number of families is 16 do we come across instances where the algorithm struggles a lot to find the optimal solution. For these difficult instances, when the computational burden of the B&B algorithm starts to become excessive, the heuristic gives quite good upper bounds that are on the average within 1.6% of the optimum.

Concluding remarks

We have presented a B&B algorithm for minimizing the sum of completion times in a single-machine scheduling setting with sequence-dependent family setup times. The B&B algorithm employs a new lower bounding scheme that is based on a network formulation of the problem. The network representation of the problem can be considered as a natural extension of the dynamic programming approaches to the problem. The main contribution of our work is in the development of network reduction rules and lower and upper bounding procedures that have enabled us to efficiently solve relatively large-size problems. With extensive computational tests, we demonstrate that the B&B algorithm can solve problems with up to 60 jobs and 12 families, where setup and processing times are uniformly distributed in various combinations of the [1,50] and [1,100] ranges.

References

1 Webster S and Baker KR (1995). Scheduling groups of jobs on a single machine. *Opns Res* **43**: 692–703.
2 Ahn BH and Hyun JH (1990). Single facility multi-class job scheduling. *Comput Opns Res* **17**: 265–272.
3 Mason AJ and Anderson EJ (1991). Minimizing flow time on a single machine with job classes and setup times. *Naval Res Logist* **38**: 333–350.
4 Potts CN and Van Wassenhove LN (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *J Opl Res Soc* **43**: 395–406.
5 Ernst AT, Krishnamoorthy M and Storer RH (1999). Heuristic and exact algorithms for scheduling aircraft landings. *Networks* **34**: 229–241.
6 Moss S, Dale C and Brame G (2000). Sequence-dependent scheduling at Baxter International. *INTERFACES* **30**: 70–80.
7 Schaller EJ, Gupta JND and Vakharia AJ (2000). Scheduling a flowline manufacturing cell with sequence dependent family setup times. *Eur J Opl Res* **125**: 324–339.
8 Lawler EL, Lenstra JK, Rinnooy Kan AHG and Shmoys DB (1993). Sequencing and scheduling: algorithms and complexity. In: *Handbooks in Operations Research and Management Science* Vol 4. Elsevier: Amsterdam, 1993. Q1
9 Rinnooy Kan AHG (1976). *Machine Scheduling Problems: Classification; Complexity and Computations*. Nijhoff: The Hague.
10 Gupta JND (1988). Single facility scheduling with multiple job classes. *Eur J Opl Res* **33**: 42–45. Q2
11 Monma CL and Potts CN (1989). On the complexity of scheduling with batch setup times. *Opns Res* **37**: 798–804.
12 Ghosh JB (1994). Batch scheduling to minimize total completion time. *Opns Res Lett* **16**: 271–275.
13 Crauwels HAJ, Hariri AMA, Potts CN and Van Wassenhove LN (1998). Branch and bound algorithms for single machine scheduling with batch setup times to minimize total weighted completion time. *Ann Opns Res* **83**: 59–76.
14 Dunstall S, Wirth A and Baker K (2000). Lower bounds and algorithms for flowtime minimization on a single machine with set-up times. *J Scheduling* **3**: 51–69.
15 Mason AJ (1992). *Genetic Algorithms and Scheduling Problems*. PhD thesis, University of Cambridge.
16 Crauwels HAJ, Potts CN and Van Wassenhove LN (1997). Local search heuristics for single machine scheduling with batch setup times to minimize total weighted completion time. *Ann Opns Res* **70**: 261–279.
17 Fisher ML (1985). An applications oriented guide to lagrangean relaxation. *Interfaces* **15**: 10–21.