Exact algorithms for the job sequencing and tool switching problem

GILBERT LAPORTE , JUAN JOSÉ SALAZAR-GONZÁLEZ & FRÉDÉRIC SEMET

IIE Transactions (2004) **36**, 37–45
Copyright © "IIE"
ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170490257871

Exact algorithms for the job sequencing and tool switching problem

GILBERT LAPORTE[1], JUAN JOSÉ SALAZAR-GONZÁLEZ[2] and FRÉDÉRIC SEMET[3]

[1]*Centre de Recherche sur les transports, Université de Montréal, C.P. 6128, Succ. Centre-ville, Montréal, Canada H3C 3J7*
E-mail: gilbert@crt.umontreal.ca
[2]*Departamento de Estadística, Investigación Operativa y Computacion, Facultad de Matemáticas, Universidad de La Laguna, 38271 La Laguna, Tenerife, Spain*
E-mail: jjsalaza@ull.es
[3]*LAMIH, Bureau 85-Bâtiment ISTV 2, Université de Valenciennes et du Hainaut-Cambrésis, Le Mont Houy-59313 Valenciennes Cedex 9, France*
E-mail: frederic.semet@univ-valenciennes.fr

Received August 2002 and accepted May 2003

The *job Sequencing and tool Switching Problem* (SSP) involves optimally sequencing jobs and assigning tools to a capacitated magazine in order to minimize the number of tool switches. This article analyzes two integer linear programming formulations for the SSP. A branch-and-cut algorithm and a branch-and-bound algorithm are proposed and compared. Computational results indicate that instances involving up to 25 jobs can be solved optimally using the branch-and-bound approach.

1. Introduction

The *job Sequencing and tool Switching Problem* (SSP) arises in the following flexible manufacturing context. We are given a set of jobs $J = \{1, \ldots, n\}$ to be processed sequentially without preemption on a single flexible machine, and a set of tools $T = \{1, \ldots, m\}$. Denote by J_t the set of jobs requiring tool t. Each job j requires a subset of tools T_j to be loaded on a magazine capable of holding at most c tools at a time, where $c \geq \max_j\{|T_j|\}$. In most practical situations, the magazine cannot hold all tools at once, so that some tool switches may be necessary when performing two jobs in succession. A tool switch consists of removing a tool from the magazine and inserting another one in its place. The order of the tools in the magazine is irrelevant. The SSP consists of simultaneously determining the processing sequence of jobs and the subsets of tools present in the magazine for each job in order to minimize the total number of tool switches. In some cases instance size can be reduced through the application of the following dominance rule (Tang and Denardo, 1988): if two jobs i and j are such that $T_i \subseteq T_j$, then job i can be eliminated since it is never suboptimal to execute it immediately after j without any tool change.

Belady (1966), McGeoch and Sleator (1991) and Privault and Finke (2000) describe applications of the SSP arising in computer memory management. The problem arises when pages (tools) have to be transferred from a slow memory to a fast memory (tool magazine) in order to execute jobs. Several variants and extensions of the SSP are surveyed in Blazewicz and Finke (1994), in Crama (1997) and in Balakrishnan and Chakravarty (2001).

Oerlemans (1992) and Crama *et al.* (1994) have provided a formal proof that the SSP is NP-hard for $c \geq 2$. Also, minimizing the number of tool switches for a given job sequence can be solved in polynomial-time by means of a Keep Tool Needed Soonest (KTNS) policy (Bard, 1988; Tang and Denardo, 1988). This policy states that when tool changes are necessary, the tools required the soonest for an upcoming job should be kept first in the magazine. Blazewicz and Finke (1994) mention that the KTNS property was already known to Belady (1966). Several heuristics have been proposed for the SSP (Bard, 1988; Kiran and Krason, 1988; Tang and Denardo, 1988; Oerlemans, 1992; Gray *et al.*, 1993; Hertz and Widmer, 1993; Crama *et al.*, 1994; Follonier, 1994; Sodhi *et al.*, 1994; Avci and Akturk, 1996; Hertz *et al.*, 1998; Privault and Finke, 2000; Knuutila *et al.*, 2002). Tang and Denardo (1988) have proposed an Integer Linear Programming (ILP) formulation of the SSP which yields poor computational results when solved by an LP-based scheme. One reason for this poor performance is, as we will show, that the value of the linear relaxation is always zero. No other model or exact algorithm is known to exist for the SSP.

The contribution of this article is to propose a new ILP formulation having a better linear relaxation than that of Tang and Denardo (1988) as well as two exact algorithms.

The first is an LP-based branch-and-cut algorithm that solves the new formulation, and the second is a direct branch-and-bound approach that does not use LP. The remainder of this article is organized as follows. In Section 2, two ILP formulations for the SSP are presented. This is followed by the branch-and-cut and branch-and-bound algorithms in Section 3, by computational results in Section 4, and by conclusions in Section 5.

2. ILP formulation

We describe two different ILP formulations for the SSP and we show that the second dominates the first relative to the value of the linear relaxation.

2.1. *The model of Tang and Denardo (1988)*

The Tang and Denardo (1988) formulation uses three families of zero-one variables. Binary variables u_{jk} are equal to one if and only if job $j \in J$ is processed in position $k \in K := \{1, \dots, n\}$. Binary variables v_{kt} are equal to one if and only if tool $t \in T$ is present in the magazine while a job is being processed in position k. Finally, binary variables w_{kt} are equal to one if and only if tool t is in the magazine while performing a job in position k but is absent from the magazine for the job processed in position $k-1$, where $k \in K \setminus \{1\}$. In other words, $w_{kt} = 1$ corresponds to a tool switch. The formulation is then:

$$\text{(TD)} \quad \min \sum_{k \in K \setminus \{1\}} \sum_{t \in T} w_{kt}, \tag{1}$$

subject to

$$\sum_{j \in J} u_{jk} = 1 \qquad (k \in K), \tag{2}$$

$$\sum_{k \in K} u_{jk} = 1 \qquad (j \in J), \tag{3}$$

$$\sum_{j \in J_t} u_{jk} \le v_{kt} \qquad (k \in K \text{ and } t \in T), \tag{4}$$

$$\sum_{t \in T} v_{kt} \le c \qquad (k \in K), \tag{5}$$

$$v_{kt} - v_{k-1,t} \le w_{kt} \qquad (k \in K \setminus \{1\} \text{ and } t \in T), \tag{6}$$

$$u_{jk} \in \{0, 1\} \qquad (k \in K \text{ and } j \in J), \tag{7}$$

$$v_{kt} \in \{0, 1\} \qquad (k \in K \text{ and } t \in T), \tag{8}$$

$$w_{kt} \in \{0, 1\} \qquad (k \in K \text{ and } t \in T). \tag{9}$$

In this formulation, the objective function minimizes the total number of tool switches while constraints (2) and (3) are standard assignment constraints. Constraints (4) mean that no job requiring tool t can be processed in position k unless the appropriate tools are present in the magazine. Constraints (5) guarantee that the magazine capacity is never exceeded. Note that these constraints can be stated as equalities since there is no need to remove unnecessary tools from the magazine. Constraints (6) mean that a tool switch must be counted whenever a tool is present in position k but was not present in position $k-1$. The number of variables in this model is $n^2 + 2nm$ and the number of constraints is $3n + 2nm - m$.

Tang and Denardo (1988) have conducted experiments with this formulation and have obtained rather disappointing results. This is hardly surprising since the linear relaxation value of the (TD) formulation (when no job has been fixed) is always equal to zero. Indeed the solution $u_{jk} = 1/n$ for all $j \in J$ and $k \in K$, $v_{kt} = |J_t|/n$ for all $k \in K$ and $t \in T$, and $w_{kt} = 0$ for all $k \in K$ and $t \in T$ is optimal for the linear relaxation. A consequence of this is that solving the SSP using this model requires almost complete enumeration.

2.2. *A new model*

We propose an alternative formulation using different variables yielding a better linear relaxation. The model exploits the fact that the job sequencing component of the SSP corresponds to a tour as long as a *dummy job* is introduced. Hence a *Travelling Salesman Problem* (TSP)-based formulation can be used. Let a zero denote a dummy job representing the start and end of operations. Binary variables x_{ij} are equal to one if and only if job i is immediately followed by job j in the sequence ($i, j \in J \cup \{0\}$). Binary variables y_{it} are equal to one if and only if tool t is in the magazine while job i is being processed ($i \in J$, $t \in T$). Binary variables z_{it} are equal to one if and only if tool t is inserted in the magazine at the start of processing of job i. In other words, $z_{it} = 1$ corresponds to a tool switch. Our model is then:

$$\text{(LSS)} \quad \min \sum_{i \in J} \sum_{t \in T_i} z_{it}, \tag{10}$$

subject to

$$\sum_{j \in J \cup \{0\} \setminus \{i\}} x_{ij} = 1 \qquad (i \in J \cup \{0\}), \tag{11}$$

$$\sum_{i \in J \cup \{0\} \setminus \{j\}} x_{ij} = 1 \qquad (j \in J \cup \{0\}), \tag{12}$$

$$\sum_{i,j \in S} x_{ij} \le |S| - 1 \qquad (S \subset J \cup \{0\} \text{ and } 2 \le |S| \le n-1), \tag{13}$$

$$\sum_{t \in T} y_{jt} \le c \qquad (j \in J), \tag{14}$$

$$x_{ij} + y_{jt} - y_{it} \le z_{jt} + 1 \qquad (i \in J \cup \{0\}, j \in J \text{ and } t \in T), \tag{15}$$

$$y_{it} = 1 \qquad (i \in J \text{ and } t \in T_i), \tag{16}$$

$$z_{it} = 0 \qquad (i \in J \text{ and } t \in T \setminus T_i), \tag{17}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j \in J \cup \{0\}), \tag{18}$$

$$y_{it} \in \{0, 1\} \qquad (i \in J \text{ and } t \in T), \tag{19}$$

$$z_{it} \in \{0, 1\} \qquad (i \in J \text{ and } t \in T). \tag{20}$$

In this model, the objective function minimizes the number of tool switches while constraints (11) and (12) are assignment constraints. Constraints (13) prevent the formation of subtours. Constraints (14) are capacity

Table 1. Tang and Denardo example: tools used in each job and $c = 4$. The original example contained 10 jobs, four of which were eliminated through the dominance rule

	Jobs					
	1	*2*	*3*	*4*	*5*	*6*
Tools	1	1	2	1	3	1
	4	3	6	5	5	2
	8	5	7	7	8	4
	9		8	9		

constraints. Constraints (15) can be explained as follows. If tool t was not in the magazine for job i and was not inserted for j, then job j cannot follow job i and require tool t at the same time. Constraints (17) ensure that a tool switch is only performed when the tool is required to immediately execute a job. The number of variables in (LSS) is $O(n^2 + nm)$, while the number of constraints is exponential as is the case of TSP models that include the Dantzig, Fulkerson and Johnson subtour elimination constraints (13) (see Dantzig *et al.*, 1954).

Using the example given by Tang and Denardo (1988) (see Table 1), we found that the optimal value of the linear relaxation of (LSS) including the subtour elimination constraints is equal to 1.9. In other words, the first formulation is dominated by the second one with respect to the value of the linear relaxation: it is always zero in (TD) but can be positive in our model.

Several constraints can be added to (LSS) to increase the value of its linear relaxation. First observe that if $|T_{i^*}| = c$ for some $i^* \in J$, then any job j immediately following i^* will generate $|T_j \setminus T_{i^*}|$ tool switches and therefore $z_{jt} \geq x_{i^*j}$ for all $j \neq i^*$ and $t \in T_j \setminus T_{i^*}$. A term equal to $\sum_{j \neq i^*} |T_j \setminus T_{i^*}| x_{i^*j}$ can then be added to the objective function. With these changes, constraints (15) must be eliminated for i^*, all $j \neq i^*$ and $t \in T_j \setminus T_{i^*}$. The objective function of LSS is therefore replaced with the following lifted objective:

$$\min \sum_{i \in J} \sum_{t \in T_i} z_{it} + \sum_{i \in J : |T_i| = c} \sum_{j \neq i} |T_j \setminus T_i| x_{ij}. \qquad (21)$$

Note that if $|T_j| = c$ for all $j \in J$ then (LSS) reduces to a TSP with costs:

$$l_{ij} := \max\{0, |T_i \cup T_j| - c\},$$

as observed by Hertz *et al.* (1998). Crama *et al.* (1994) have shown that the value of l_{ij} is a lower bound on the number of tool switches when j immediately follows i. If $|T_i| = c$ for all $i \in J$, then l_{ij} is then the actual number of tool switches. To see that a TSP is obtained in this case, note that constraints (14) are all trivially satisfied as equalities and all constraints (15) are removed. Also variables z_{it} are removed from the constraints, and the objective function becomes:

$$\min \sum_{i \in J : |T_i| = c} \sum_{j \neq i} l_{ij} x_{ij}. \qquad (22)$$

Objective functions (10) and (21) are equivalent at an integer solution, but since the first double summation is the same in these two objectives, objective function (21) may provide a non-trivial improvement at a fractional solution.

A valid inequality is obtained by computing a lower bound on the number of tool switches necessary to process a job j:

$$\sum_{t \in T_j} z_{jt} \geq \sum_{i \neq j : |T_i| \neq c} l_{ij} x_{ij} \qquad (j \in J). \qquad (23)$$

The following inequality states that either tool t is introduced at the start of job j ($z_{jt} = 1$) or j immediately follows a job i requiring t in which case $z_{jt} = 0$:

$$\sum_{i \in J_t \setminus \{j\}} x_{ij} + z_{jt} \leq 1 \qquad (t \in T,\ j \in J). \qquad (24)$$

Now suppose that j immediately follows i, and $|T_i| = c$, $|T_j| < c$. Then the following inequality states that all tools present in the magazine beyond the set T_j required for j can come from T_i. In other words, it does not pay to make unnecessary tool switches:

$$\sum_{t \in T_i \setminus T_j} y_{jt} \geq (c - |T_j|) x_{ij} \quad (i, j \in J : |T_i| = c, |T_j| < c). \qquad (25)$$

On the Tang and Denardo (1988) example, using the lifted objective (21) and introducing constraints (23), (24) and (25) in the (LSS) model increases the value of the linear relaxation to 6.0. Without the additional constraints, the value of the linear relaxation is 2.91. The optimal solution value of the model is 7.0.

3. Exact algorithms

We have developed two algorithms for the determination of an optimal solution of the SSP.

3.1. *Branch-and-cut algorithm*

The first algorithm we propose for the SSP uses branch-and-cut. This technique was first proposed for the solution of the TSP (Grötschel and Padberg, 1985) and is now used for the solution of several combinatorial optimization problems (Caprara and Fischetti, 1997). Branch-and-cut is essentially a branch-and-bound scheme that solves a linear program at each node of the search tree, gradually introducing violated constraints. Our branch-and-cut algorithm is based on the (LSS) formulation.

At the root of the search tree, a feasible solution (upper bound) is constructed as follows. A job sequence is obtained by heuristically solving a TSP with edge costs $1 - (x^*_{ij} + x^*_{ji})$, where x^*_{ij} and x^*_{ji} are the values taken by x_{ij} and x_{ji} in the LP solution. The heuristic used to solve this TSP is called GENIUS (Gendreau *et al.*, 1992). Starting with three vertices, it gradually constructs a TSP solution by inserting at each iteration a vertex in the partially constructed tour

Table 2. Tang and Denardo (1988) example: l_{ij} values

	l_{ij}					
	1	*2*	*3*	*4*	*5*	*6*
1	—	2	3	2	2	1
2		—	3	1	0	1
3			—	3	2	2
4				—	2	2
5					—	2

while performing a local reoptimization. Procedure KTNS is then applied to the TSP sequence obtained in this manner.

At each node of the search tree, the most violated subtour elimination constraints (13) (if any) are found through the application of a maximum flow algorithm, as suggested by Grötschel and Padberg (1985). At most 50 violated constraints are introduced in the program.

At a generic node of the search tree a lower bound on the optimal solution of the current subproblem is computed by solving a linear program in which constraints (13) and the integrality conditions have been relaxed, but using the modified objective (21) as well as constraints (23), (24) and (25). Whenever the solution of the strengthened linear relaxation is not feasible we look for violated subtour elimination constraints (13) by means of the classical maximum flow/minimum cut algorithm (Grötschel and Padberg, 1985).

When no violated inequality can be identified, branching is made on the x_{ij} variables so as to generate a job sequence rooted at the dummy job. In other words, we are not using a binary branching of the form "$x_{ij} = 0$ or $x_{ij} = 1$," but a broad branching of the form "$x_{i_p j} = 1$ for each $j \in Q := J \setminus \{i_1, \ldots, i_p\}$," where $i_1, \ldots, i_p$ is the current sequence of fixed jobs. To illustrate, suppose that $J = \{1, 2, 3, 4, 5\}$ and that $p = 2$ jobs have already been fixed in earlier branches: $i_1 = 5$ and $i_2 = 3$. Then $Q = \{1, 2, 4\}$ and the three branches $x_{3,1} = 1$, $x_{3,2} = 1$, and $x_{3,4} = 1$ are created, meaning that job 3 is to be followed by one of the three jobs 1, 2 or 4. We found this branching to be empirically better than the

Table 3. Computational results for the branch-and-cut algorithm on randomly generated instances with eight jobs

Jobs	*Tools*	*Min*	*Max*	*c*	*Solved*	*LB/OPT*	*UB/OPT*	*SECs*	*Nodes*	*Seconds*
8	15	2	5	5	10	0.615	1.164	111.1	5503.9	81.9
8	15	2	5	10	10	0.777	1.287	216.2	35 409.3	371.2
8	15	5	10	10	10	0.749	1.245	97.9	3917.4	72.1
8	15	2	10	10	10	0.728	1.365	132.4	7382.9	99.7
8	20	2	5	5	10	0.595	1.166	121.2	9564.9	178.2
8	20	2	5	10	10	0.624	1.219	232.2	50 006.5	775.7
8	20	5	10	10	10	0.639	1.262	125.9	5790.4	189.7
8	20	2	10	10	10	0.615	1.254	148.7	14 915.8	325.0
8	20	2	5	15	10	0.913	1.218	152.8	36 585.4	469.2
8	20	5	10	15	10	0.770	1.390	224.8	23 799.9	404.6
8	20	10	15	15	10	0.795	1.193	75.8	2131.9	47.0
8	20	2	10	15	10	0.791	1.342	219.0	31 119.1	476.8
8	20	5	15	15	10	0.723	1.405	185.8	9014.8	222.6
8	20	2	15	15	10	0.737	1.325	169.7	10 206.2	248.3
8	25	2	5	5	10	0.593	1.117	131.3	12 169.4	252.0
8	25	2	5	10	10	0.569	1.197	236.1	52 531.5	1055.6
8	25	5	10	10	10	0.692	1.199	103.0	5447.8	202.0
8	25	2	10	10	10	0.535	1.241	171.8	16 854.5	580.0
8	25	2	5	15	10	0.854	1.194	223.4	52 700.2	928.0
8	25	5	10	15	10	0.616	1.374	239.3	32 057.8	1057.3
8	25	10	15	15	10	0.878	1.166	45.7	1226.2	46.2
8	25	2	10	15	10	0.658	1.317	235.8	43 993.8	1037.3
8	25	5	15	15	10	0.602	1.379	175.8	13 533.6	572.3
8	25	2	15	15	10	0.604	1.340	204.6	22 865.3	816.7
8	25	2	5	20	10	1.000	1.066	3.4	13.7	0.8
8	25	5	10	20	10	0.828	1.397	226.0	42 120.2	684.9
8	25	10	15	20	10	0.768	1.362	228.9	14 394.6	443.2
8	25	15	20	20	10	0.911	1.172	35.1	614.8	11.4
8	25	2	10	20	10	0.866	1.320	199.9	42 367.1	693.0
8	25	5	15	20	10	0.803	1.398	223.4	23 426.0	502.0
8	25	10	20	20	10	0.745	1.257	143.6	7615.7	209.0
8	25	2	15	20	10	0.810	1.380	225.4	35 652.1	638.1
8	25	5	20	20	10	0.760	1.312	155.3	8437.1	235.0
8	25	2	20	20	10	0.778	1.311	184.1	16 092.0	373.6

standard binary branching. The search tree is explored in a depth-first fashion.

3.2. *Branch-and-bound algorithm*

We have also developed a branch-and-bound algorithm for the SSP. It makes use of an initial upper bound obtained by the application of the following *greedy heuristic*. The first job is the one requiring the largest number of tools; to break ties, the job with the most frequently used tools is selected. Then, at each iteration, the job having the largest number of tools in common with the last job is selected; to break ties, we minimize the total number of tools used by the last job in the current sequence and the job being scheduled. While more sophisticated heuristics are available for the SST (Hertz *et al.*, 1998), we did not see any advantage in refining our simple heuristic since, for the problem sizes considered, there was practically no difference in the size of the search tree, whether it was initiated with our heuristic solution or with the optimal solution.

Given a partial sequence of jobs $(i_1, \ldots, i_p)$, a lower bound on the optimal value is obtained by computing the minimum number of tool switches to process the jobs in the partial sequence (using a KTNS policy for the partial sequence) plus the maximum of the two following lower bounds:

Lower bound 1: The number of tools required by the last and forthcoming jobs, minus c. Indeed, a valid lower bound is the number of tools necessary for the forthcoming jobs and not necessary for the last job in the partial sequence, minus the number of free slots in the magazine (bounded above by the capacity of the magazine minus the tools required by the last job). In other words, the lower bound is:

$$\begin{aligned} &\left| \cup_{i\in Q} T_i \setminus T_{i_p} \right| - \left(c - \left|T_{i_p}\right|\right) \\ &\quad = \left|T_{i_p} \cup (\cup_{i\in Q} T_i)\right| - \left|T_{i_p}\right| - c + \left|T_{i_p}\right| \\ &\quad = \left|T_{i_p} \cup (\cup_{i\in Q} T_i)\right| - c. \end{aligned}$$

Table 4. Computational results for the branch-and-cut algorithm on randomly generated instances with nine jobs

Jobs	*Tools*	*Min*	*Max*	*c*	*Solved*	*LB/OPT*	*UB/OPT*	*SECs*	*Nodes*	*Seconds*
9	15	2	5	5	10	0.612	1.244	232.8	28 091.0	767.0
9	15	2	5	10	9	0.732	1.379	429.4	134 725.6	2432.1
9	15	5	10	10	9	0.750	1.265	189.2	10 334.2	383.1
9	15	2	10	10	10	0.710	1.406	276.0	37 044.3	902.5
9	20	2	5	5	9	0.602	1.151	221.2	40 273.4	1283.5
9	20	2	5	10	9	0.617	1.331	401.9	126 077.2	2825.2
9	20	5	10	10	9	0.661	1.244	241.8	21 376.3	1214.1
9	20	2	10	10	6	0.622	1.256	263.3	31 548.0	1424.2
9	20	2	5	15	9	0.904	1.356	290.2	98 304.3	1737.7
9	20	5	10	15	5	0.782	1.528	447.0	67 785.8	2415.1
9	20	10	15	15	10	0.769	1.264	167.1	9045.2	366.6
9	20	2	10	15	4	0.811	1.420	401.3	129 573.5	2532.0
9	20	5	15	15	9	0.703	1.392	348.1	43 409.3	1886.8
9	20	2	15	15	8	0.703	1.446	355.0	41 242.5	1948.2
9	25	2	5	5	9	0.612	1.135	253.6	47 458.3	1688.5
9	25	2	5	10	5	0.548	1.215	377.2	117 100.4	2976.3
9	25	5	10	10	9	0.679	1.216	202.9	24 348.3	1601.9
9	25	2	10	10	1	0.519	1.296	285.0	50 494.0	3178.6
9	25	2	5	15	9	0.814	1.230	353.6	118 109.2	2731.3
9	25	10	15	15	10	0.866	1.219	99.6	5397.9	349.4
9	25	2	10	15	1	0.652	1.391	401.0	120 895.0	3459.9
9	25	5	15	15	4	0.612	1.414	310.8	41 433.5	2701.9
9	25	2	15	15	1	0.600	1.240	368.0	116 744.0	3267.5
9	25	2	5	20	10	1.000	1.215	8.3	42.0	1.7
9	25	5	10	20	7	0.814	1.530	407.6	125 601.0	2925.8
9	25	10	15	20	1	0.741	1.444	486.0	50 700.0	3095.0
9	25	15	20	20	10	0.888	1.182	81.8	3078.9	97.7
9	25	2	10	20	9	0.861	1.445	348.2	111 075.9	2526.5
9	25	5	15	20	3	0.811	1.513	447.0	80 084.0	2812.4
9	25	10	20	20	10	0.733	1.297	286.1	32 740.4	1588.9
9	25	2	15	20	4	0.800	1.490	405.3	125 347.8	3086.1
9	25	5	20	20	8	0.726	1.328	296.0	35 628.6	1779.6
9	25	2	20	20	5	0.746	1.360	303.8	54 889.0	2016.8

Lower bound 2: The minimum cost of an i_p-spanning tree on the vertices of $\{i_p\} \cup Q$. (An i_p-spanning tree is a spanning tree on Q, plus the least cost edge connecting it to i_p.) Here the cost of an edge (i, j) is equal to l_{ij}. This bound uses the fact that the jobs of $\{i_p\} \cup Q$ form a Hamiltonian chain.

The computational complexity of both lower bounds is polynomial in n and m. Indeed, the first lower bound can be computed in $O(m)$ time, while the second takes $O(n^2 \log n)$ time if Kruskal's algorithm (Kruskal, 1956) is used to compute the spanning tree. We take the maximum of these two lower bounds since there is no dominance between them. To illustrate this observation, let us consider the Tang and Denardo example (see Table 1). First, assume that the partial sequence consists solely of job 2. Then, the value of lower bound 1 is equal to five ($= (n-1) - c = 9 - 4$) whereas the value of lower bound 2 is seven. The last value can be obtained by considering the following 2-spanning tree: $\{(2, 5), (5, 1), (1, 6), (6, 3), (6, 4)\}$ (see Table 2). Next, assume that the partial sequence is 1, 3, 4. The value of lower bound 1 is equal to three ($= (n-3) - c = 7 - 4$) and the value of lower bound 2 is two since the 4-spanning tree is $\{(4, 2), (2, 5), (2, 6)\}$ (see Table 2).

The branching rule is to consider jobs in the order in which they appear in the greedy algorithm. In other words, the next job selected for branching is the first job of the greedy order which has not previously been fixed in the branching, using recursivity. For example, if the greedy sequence is (5,3,4,1,2), then the first branch explored in the tree will be 5-3-4-1-2, and the second branch will be 5-3-4-2-1. The second branch emanating immediately from job "3" will be 5-3-1-4-2, etc. Branching is again performed in a depth-first manner. Also the upper bound is updated at

Table 5. Computational results for the branch-and-bound algorithm on randomly generated instances with 15 jobs

					LB1			*LB1 + LB2*		
Jobs	*Tools*	*Min*	*Max*	*c*	*Solved*	*Nodes*	*Seconds*	*Solved*	*Nodes*	*Seconds*
15	15	2	5	5	10	6169 868.8	15.5	10	872 889.9	6.3
15	15	2	5	10	10	203.1	0.0	10	203.1	0.0
15	15	5	10	10	10	238 466 159.8	1151.3	10	12 695 066.5	146.5
15	15	2	10	10	10	57 740 124.6	230.2	10	3248 709.3	38.6
15	20	2	5	5	10	13 028 793.0	38.8	10	4015 611.7	30.6
15	20	2	5	10	10	236.8	0.0	10	236.8	0.0
15	20	5	10	10	9	121 528 400.6	715.4	10	6973 355.3	94.0
15	20	2	10	10	9	144 053 951.7	635.1	10	54 641 596.0	647.3
15	20	2	5	15	10	120.0	0.0	10	120.0	0.0
15	20	5	10	15	10	925 615.8	2.7	10	925 615.8	10.7
15	20	10	15	15	8	189 632 765.3	1340.6	10	1995 247.1	41.0
15	20	2	10	15	10	1494 274.5	3.7	10	1494 274.5	15.5
15	20	5	15	15	9	193 574 251.1	944.8	10	45 984 478.8	669.9
15	20	2	15	15	9	146 527 113.2	796.7	9	24 426 805.0	394.4
15	25	2	5	5	10	6800 524.6	20.7	10	2934 256.2	23.3
15	25	2	5	10	10	140.3	0.0	10	140.3	0.0
15	25	5	10	10	8	145 152 001.4	891.8	10	25 601 323.3	385.1
15	25	2	10	10	10	89 296 250.9	479.4	10	28 979 040.4	371.2
15	25	2	5	15	10	120.0	0.0	10	120.0	0.0
15	25	5	10	15	10	1340 612.1	4.5	10	1324 492.4	18.6
15	25	10	15	15	5	240 148 770.0	2224.5	10	3376 786.9	80.5
15	25	2	10	15	10	609 674.0	1.7	10	609 674.0	6.9
15	25	5	15	15	8	194 139 754.6	1181.1	8	54 864 983.3	915.5
15	25	2	15	15	9	70 493 769.2	349.2	10	9713 799.6	134.4
15	25	2	5	20	10	120.0	0.0	10	120.0	0.0
15	25	5	10	20	10	40 618.1	0.1	10	40 618.1	0.4
15	25	10	15	20	10	8668 165.1	44.5	10	7822 686.2	168.8
15	25	15	20	20	1	292 479 934.0	2713.4	10	1933 537.9	58.5
15	25	2	10	20	10	139.1	0.0	10	139.1	0.0
15	25	5	15	20	10	9559 616.3	33.7	10	9559 616.3	139.0
15	25	10	20	20	7	253 697 116.4	1897.0	10	29 057 947.8	700.5
15	25	2	15	20	10	9713 799.6	34.6	10	9713 799.6	134.4
15	25	5	20	20	7	287 060 797.3	1859.3	9	52 172 340.4	1037.8
15	25	2	20	20	8	225 726 489.0	1497.1	8	68 791 747.2	1336.9

Table 6. Computational results for the branch-and-bound algorithm with LB1 + LB2 on randomly generated instances with 20 and 25 jobs

Jobs	*Tools*	*Min*	*Max*	*c*	*Solved*	*Nodes*	*Seconds*
20	15	2	5	5	8	131 899 592.2	1375.0
20	15	2	5	10	9	1671 881.2	14.9
20	15	2	10	10	1	15 334 759.0	170.4
20	20	2	5	5	6	86 609 305.0	929.8
20	20	2	5	10	10	25 376 623.1	225.0
20	20	2	5	15	10	221.2	0.0
20	20	5	10	15	4	110 670 505.0	1771.9
20	20	2	10	15	4	23 796 204.5	349.3
20	25	2	5	5	3	116 304 539.3	1226.6
20	25	2	5	10	10	17 861 308.4	170.2
20	25	2	5	15	10	225.5	0.0
20	25	5	10	15	1	33 106 709.0	536.3
20	25	10	15	15	2	39 590 854.0	1432.4
20	25	2	10	15	5	67 379 248.8	1025.4
20	25	2	5	20	10	210.0	0.0
20	25	5	10	20	8	16 763 635.6	274.2
20	25	2	10	20	10	42837.6	0.5
20	25	5	15	20	1	146 490 505.0	3094.2
25	15	2	5	10	4	9863 072.2	84.1
25	20	2	5	10	1	187 936 788.0	2284.9
25	20	2	10	15	1	7458 838.0	102.5
25	25	2	5	10	3	7320 014.0	73.9
25	25	2	5	15	10	895.3	0.0
25	25	2	5	20	10	332.8	0.0
25	25	5	10	20	1	688 098.0	10.9
25	25	2	10	20	8	4725 531.6	70.6

every node of the search tree: the partial sequence is simply extended by applying the initial greedy rule.

4. Computational results

The algorithms just described were coded in C and run on a Sunfire 4800 Computer (900 MHz). The linear programs were solved using CPLEX 7.1. A maximum of 3600 seconds was imposed on the solution time of any instance.

We first tested the branch-and-cut algorithm on instances generated as in Crama *et al.* (1994). In this generation scheme, the minimum and the maximum number of tools required for any job are given. For each job j, the number of tools $|T_j|$ is then randomly generated within this interval and the set T_j is randomly generated while ensuring that no set of tools is included in any other. Therefore, the dominance rule described at the beginning of Section 1 never applies to these instances. Results are reported in Tables 3 and 4 under the following headings:

- Jobs: number of jobs ($|J| = n$).
- Tools: number of tools ($|T| = m$).
- c: magazine capacity.
- Min: minimum number of tools required for a job.
- Max: maximum number of tools required for a job.
- Solved: number of instances out of 10 that could be solved within 3600 seconds.
- LB/OPT: ratio of the lower bound value over the optimum at the root of the search tree.
- UB/OPT: ratio of the upper bound value over the optimum at the root of the search tree.
- SECs: total number of subtour elimination constraints (13) generated in the branch-and-cut algorithm.
- Nodes: number of nodes in the search tree.
- Seconds: computation time in seconds.

The reported statistics are averages over the number of solved instances. Our branch-and-cut algorithm was capable of solving instances containing up to nine jobs but had a very low success rate for $n \geq 10$. Both the lower and upper bounds at the root of the search tree are weak. The upper bound is not a real drawback since it is typically equal to the optimum after a few branching nodes. This is why we did not find it useful to improve it. Even if the branch-and-cut process is initiated with the optimum, it takes just as long to solve the problem. While our formulation is stronger than that of Tang and Denardo (1988), it only solves very small instances. On the positive side, this formulation works better on instances that are close to a TSP, i.e., when the minimum number of tools for a job is close to the magazine capacity. This is not surprising since, even if the TSP is a difficult combinatorial problem, in practice branch-and-cut approaches have been successfully applied to medium sized instances. When the number of tools is small compared to

Table 7. Computational results for the branch-and-bound algorithm on the first eight instance types from Hertz *et al.* (1998)

					LB1			*LB1 + LB2*		
Jobs	*Tools*	*Min*	*Max*	*c*	*Solved*	*Nodes*	*Seconds*	*Solved*	*Nodes*	*Seconds*
10	10	2	4	4	10	9015.8	0.0	10	2545.7	0.0
10	10	2	4	5	10	2478.6	0.0	10	2397.3	0.0
10	10	2	4	6	10	1413.5	0.0	10	1413.5	0.0
10	10	2	4	7	10	61.3	0.0	10	61.3	0.0
15	20	2	6	6	10	10 515 945.2	31.0	10	2654 120.8	21.8
15	20	2	6	8	10	1390 413.7	3.3	10	1355 251.7	10.7
15	20	2	6	10	10	335 957.7	0.8	10	335 957.7	2.5
15	20	2	6	12	10	214.4	0.0	10	214.4	0.0

the magazine capacity, the SSP becomes very different from the TSP, and it is therefore more difficult to solve.

We then tested the branch-and-bound algorithm on similar instances, including randomly generated instances, and eight instance types solved heuristically by Hertz *et al.* (1998). The results are presented in Tables 5, 6 and 7. The column headings are the same as in Table 3, except for LB1 and LB1 + LB2 which respectively refer to use of the first lower bound and of the maximum of the two lower bounds described in Section 3.2. Our results clearly show the advantage of using lower bound 2. Again SSP instances close to a TSP are relatively easier to solve with LB1 + LB2. With this algorithm we solved larger instances (containing 15, 20 and 25 jobs) within relatively short computing times.

It should be noted that our test problems may be more difficult to solve than what is observed in practice since no job clustering is assumed as is sometimes the case in real situations (Posner, 1986).

5. Conclusions

We have compared some formulations and exact algorithms for the SSP. We have proposed an ILP formulation that improves that of Tang and Denardo (1988). We have also devised two enumeration algorithms, the first based on branch-and-cut, the second based on branch-and-bound. Instances up to 25 jobs were solved to optimality using our best algorithm.

As already observed by Tang and Denardo (1988), the SSP is a very difficult combinatorial optimization problem for which strong lower bounds are hard to derive, irrespective of the used algorithm. Our results indicate that a traditional branch-and-bound approach seems superior to mathematical programming for this type of problem.

Acknowledgements

This research was partly supported by the Canada Research Chair in Distribution Management, by the Canadian Natural Sciences and Engineering Research Council under grant OGP00039682, by the Ministerio de Ciencia y Tecnología, Spain, and by the Nord-Pas-de-Calais region, France. This support is gratefully acknowledged. Thanks are also due to Éric Duchenne for his help with programming, to the reviewers for their valuable comments, and to Jacek Blazewicz for pointing out some useful references.

References

Avci, S. and Akturk, M.S. (1996) Tool magazine arrangement and operations sequencing on CNC machines. *Computers & Operations Research*, **23**, 1069–1081.

Balakrishnan, N. and Chakravarty, A.K. (2001) Opportunistic retooling of a flexible machine subject to failure. *Naval Research Logistics*, **48**, 79–97.

Bard, J.F. (1988) A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, **20**, 382–391.

Belady, L.A. (1966) A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, **5**, 78–101.

Blazewicz, J. and Finke, G. (1994) Scheduling with resource management in manufacturing systems. *European Journal of Operational Research*, **76**, 1–14.

Caprara, A. and Fischetti, M. (1997) Branch and cut algorithms, in *Annotated Bibliographies in Combinatorial Optimization*, Dell'Amico, M., Maffioli, F. and Martello, S. (eds.), Wiley, Chichester, UK, pp. 45–63.

Crama, Y. (1997) Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, **99**, 136–153.

Crama, Y., Kolen, A.W.J., Oerlemans, A.G. and Spieksma, F.C.R. (1994) Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, **6**, 33–54.

Dantzig, G.B., Fulkerson, D.R. and Johnson, S.M. (1954) Solution of a large-scale traveling salesman problem. *Operations Research*, **2**, 393–410.

Follonier, J.-P. (1994) Minimization of the number of tool switches on a flexible machine. *Belgian Journal of Operations Research, Statistics and Computer Science*, **34**, 55–72.

Gendreau, M., Hertz, A. and Laporte, G. (1992) New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, **40**, 1086–1094.

Gray, A.E., Seidmann, A. and Stecke, K.E. (1993) A synthesis of decision models for tool management in automated manufacturing. *Management Science*, **39**, 549–567.

Grötschel, M. and Padberg, M.W. (1985) Polyhedral Theory, in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys D.B. (eds.), Wiley, Chichester, UK, pp. 251–305.

Hertz, A., Laporte, G., Mittaz, M. and Stecke, K.E. (1998) Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions*, **30**, 689–694.

Hertz, A. and Widmer, M. (1993) An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, **65**, 319–345.

Kiran, A.S. and Krason, R.J. (1988) Automated tooling in a flexible manufacturing system. *Industrial Engineering*, **20**, 52–57.

Knuutila, T., Hirvikorpi, M., Johnsson, M. and Nevalainen, O. (2002) Grouping PCB assembly jobs with typed component feeder units. Technical report 460, Turku Centre for Computer Science, Finland.

Kruskal, J.B. (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, **7**, 48–50.

McGeoch, L.A. and Sleator, D.D. (1991) A strongly competitive randomized paging algorithm. *Algorithmica*, **6**, 816–825.

Oerlemans, A.G. (1992) Production planning for flexible manufacturing systems. Ph.D. Dissertation, University of Limburg, Maastricht, Holland.

Posner, M.E. (1986) A sequencing problem with release dates and clustered jobs. *Management Science*, **32**, 731–738.

Privault, C. and Finke, G. (2000) k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, **96**, 255–269.

Sodhi, M.S., Askin, R.G. and Sen, S. (1994) Multiperiod tool and production assignment in flexible manufacturing systems. *International Journal of Production Research*, **32**, 1281–1294.

Tang, C.S. and Denardo, V. (1988) Models arising from a flexible manufacturing machine. Part I: minimizing the number of tool switches. *Operations Research*, **36**, 767–777.

Biographies

Gilbert Laporte obtained his Ph.D. in Operations Research at the London School of Economics in 1975. He is Professor of Operations Research at HEC Montréal, Director of the *Canada Research Chair in Distribution Management*, and adjunct Professor at the University of Alberta. He is also a member of the Centre for Research on Transportation (serving as director from 1987 to 1991), founding member of the Groupe d'études et de recherche en analyse des décisions (GERAD) and Fellow of the Center for Management of Operations and Logistics, University of Texas at Austin. He has authored or coauthored several books, as well as more than 200 scientific articles in combinatorial optimization, mostly in the areas of vehicle routing, location and timetabling. He has also made more than 600 scientific presentations. He is the Editor of *Computers & Operations Research*, the past Editor of *Transportation Science* (1995–2002), and associate editor or editorial board member of several other operations research journals. He has received many scientific awards including the Pergamon Prize (United Kingdom) in 1987, the 1994 Merit Award of the Canadian Operational Research Society, the CORS Practice Prize on two occasions. In 1999, he obtained the ACFAS Jacques-Rousseau Prize for Interdisciplinarity, and the President's Medal (Operational Research Society, United Kingdom). In 2001, he was awarded the Pedagogy Prize by HEC Montréal. He has been a member of the Royal Society of Canada since 1998.

Juan José Salazar González is an Associate Professor of Mathematical Programming, "Departamento de Estadística, Investigación Operativa y Computación," University of La Laguna. His research interests include the improvement of modelling and solution techniques for the solution of combinatorial optimization problems. He currently conducts research in transportation, telecommunication and statistical disclosure control.

Frédéric Semet received his Doctorate in Science from the Department of Mathematics of the École Polytechnique Fédérale de Lausanne in 1993. He is Professor at the University of Valenciennes, France and adjunct Professor at the University of Montreal. He heads the Operations Research Group of the Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH-ROI). His research interests lie in the field of combinatorial optimization, mostly in the area of vehicle routing, location, and work force scheduling. He has been involved in a variety of projects for distribution companies.

Contributed by the Scheduling/Production Planning/Capacity Planning Departments