Baltzer Journals

A new lower bound for the Open-Shop problem

CHRISTELLE GUÉRET[1] AND CHRISTIAN PRINS[1*]

[1] *École des Mines de Nantes*
Département Automatique-Productique
La Chantrerie, 4 Rue Alfred Kastler
F-44307 Nantes Cedex 03, FRANCE
E-mail: `Christelle.Gueret@emn.fr`, `Christian.Prins@emn.fr`

In this paper we present a new lower bound for the Open-Shop problem. In shop problems, a classical lower bound LB is the maximum of job durations and machine loads. Contrary to the Flow-Shop and Job-Shop problems, the Open-Shop lacks tighter bounds. For the general Open-Shop problem OS, we propose an improved bound defined as the optimal makespan of a relaxed Open-Shop problem OS_k. In OS_k, the tasks of any job may be simultaneous, except for a selected job k. We prove the NP-hardness of OS_k. However, for a fixed processing route of k, OS_k boils down to subset-sum problems which can quickly be solved via dynamic programming. From this property, we define a branch-and-bound method for solving OS_k which explores the possible processing routes of k. The resulting optimal makespan gives the desired bound for the initial problem OS. We evaluate the method on difficult instances created by a special random generator, in which all job durations and all machine loads are equal to a given constant. Our new lower bound is at least as good as LB and improves it typically by 4%, which is remarkable for a shop problem known for its rather small gaps between LB and the optimal makespan. Moreover, the computational times on a PC are quite small on average. As a by-product of the study, we determined and we propose to the research community a set of very hard Open-Shop instances, for which the new bound improves LB by up to 30%.

Keywords. Scheduling theory, Open-Shop, Lower Bound

1 Introduction

In the Open-Shop problem OS, a set J of n jobs, each consisting of m tasks or operations, must be processed on a set M of m machines. The processing times are given by a matrix $P : m \times n$, in which p_{ij} is the duration of task O_{ij} of job j, to be done on machine i. The tasks of a job can be processed in any order, but only one at a time, and preemptions are not allowed. The objective function to be minimized is the maximum completion time of tasks, or *makespan*. This problem is NP-hard for $m \geq 3$ [3], and it is even hard to approximate within a factor of 5/4 [8].

In the sequel L_i denotes the workload of machine M_i, D_j the length of job j, and D_{ij}

[*] author to whom correspondence should be addressed

the cumulated duration of job j on machines M_i to M_m (included). For a given feasible schedule, C_{ij} stands for the completion time of O_{ij}, and $C_{\max}$ for the makespan. The optimal makespan is denoted by $C^*_{\max}$.

A classical lower bound, valid for all shop problems, is the bound LB equal to the maximum of machine loads and job durations. Whereas much better bounds are available for the Flow-Shop and Job-Shop problems, LB is still, as far as we know, the only lower bound used for the general Open-Shop problem (nevertheless, some improved bounds are known for special cases, for instance the three-machine Open-Shop with a bottleneck machine [6]). The main reason is that the optimum is equal to LB in all simple relaxations which can be solved in polynomial time, contrary to shop problems with fixed routes. Here are three examples.

Firstly, if preemptions are allowed, the problem becomes polynomial [3] and the optimal makespan is always equal to LB. Secondly, the same phenomenon occurs if subproblems restricted to two jobs or two machines are considered. Thirdly, the one-machine bounds which work well for the Job-Shop can no longer be applied since they require precedence constraints to define a release date and a tail for each operation.

In a previous study [4] we showed that most Open-Shop instances produced by a random generator are easy in practice, at least when task durations are uniformly distributed in a given interval : there exist heuristics that reach LB for 90% of instances. For the remaining 10% or for hard instances obtained by specially designed generators, LB is no longer tight and a better bound is desirable.

Obviously, since the Open-Shop is NP-hard, it is possible with some efforts to find difficult instances. The first set of such problems was proposed by Taillard [7]. It contains instances selected according to their resistance to a tabu search developed by the author. However, optimal solutions are known today for all Taillard's problems up to 9×9, in particular thanks to a branch-and-bound method designed by Brucker *et al.* [1]: as for the uniformly generated instances, the optimal makespan is nearly always equal to LB. Brucker *et al.* created harder instances, with gaps $C^*_{\max}/LB$ reaching 10%, to test their branch-and-bound algorithm. Such gaps are close to the ones encountered in the other shop problems, and this has been our initial motivation to investigate new lower bounds.

In this paper, we show that it is possible to compute an improved lower bound for the general Open-Shop problem OS. This bound is equal to the optimal makespan of a relaxed problem OS_k, in which the operations of every individual job may overlap, except for a selected job k. This problem is still NP-hard, but the special case in which the sequence of the operations of job k is fixed can be solved pseudo-polynomially in $O(mn \times LB)$. Thanks to this low complexity, it is possible to design a fast Branch-and-Bound procedure for OS_k, thus providing the desired new lower bound for the initial problem OS.

Concerning its practical value, our new bound is computed very quickly despite the NP-hardness. An obvious application would be a Branch-and-Bound algorithm for the general Open-Shop, but we are still working on this subject and we just can give some preliminary results in the conclusion.

In this context, we have developed an Open-Shop problem generator, designed to create instances with large gaps between the new bound and LB (and hence, between the optimal makespan and LB). We have built a set of instances with gaps approaching 30% for sizes 4×4 to 7×7. As a comparison, the gaps between the best known makespan and LB never exceeds 7% in the problem set considered by Brucker *et al.* [1], with the exception of a gap of almost 20% for a 4×4 problem. Our instances are very hard, since the best Branch-and-Bound method available [1] fails to solve most instances of size 7×7 and larger.

The paper is organised as follows. Section 2 defines the problem OS_k and proves its NP-hardness. Section 3 presents the pseudo-polynomial algorithm for solving OS_k for a fixed sequence of job k. The Branch-and-Bound procedure for OS_k, which finds the new lower bound for OS, is described in Section 4. Section 5 contains the results of computational experiments, the description of our random generator, and the very hard instances that we have discovered.

2 The relaxed problem OS_k

In this section we first give a definition of problem OS_k. We show that its optimal makespan is a lower bound for OS, that this bound is at least as good as the classical bound LB, and that OS_k is NP-hard.

Given the general Open-Shop problem OS and a job $k \in J$, we define OS_k as the relaxed problem in which the tasks of every individual job may overlap, except for job k. In the sequel we will use N_i to denote the set of operations to be processed on M_i, other than O_{ik}. Using this notation, any operation of N_i may overlap with any operation of N_j for $i \neq j$. Our new lower bound for OS is the optimal makespan of OS_k, $C^*_{\max}(OS_k)$: any feasible schedule for OS (in particular an optimal one) remains feasible for the relaxed problem OS_k.

Note that the new problem OS_k can still be viewed as an Open-Shop problem with one *ordinary job* with m tasks (job k), and $m(n-1)$ *degenerate jobs*, each having one non-zero task only. Therefore, apart from its use in the design of our new lower bound, OS_k is interesting in itself as a new problem on the tiny border between easy and hard special cases of the Open-Shop: we shall see that this weakly constrained problem is still NP-hard.

Proposition 1
We can choose k such that the optimal makespan of OS_k gives a lower bound for OS at least as good as the classical bound LB.

Proof
Since $C^*_{\max}(OS_k) \geq max\{L_1, \ldots, L_m, D_k\}$, the job k with $D_k = max\{D_1, \ldots, D_n\}$ leads to a lower bound at least as good as LB. □

Proposition 2
Problem OS_k is binary NP-hard.

Proof
Clearly, the decision version of this problem (is there a schedule not longer than a threshold T?) belongs to NP. Consider the *PARTITION* decision problem, known to be NP-complete: given a set $S = \{a_1, a_2, \ldots, a_p\}$ of p integers with $\sum_{i=1}^{p} a_i = 2B$, is it possible

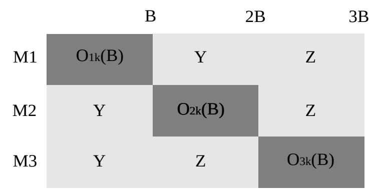

Figure 1: Solution of makespan $3B$ for IDk

to partition S into two subsets Y and Z such that $\sum_{i\in Y} a_i = \sum_{i\in Z} a_i = B$?

For any instance of *PARTITION*, we can construct in polynomial time an instance for OS_k with $m = 3$ machines and $T = 3B$, as follows:

- We define job k as an ordinary job with three tasks of duration B.
- To each integer a_i we associate a relaxed job with three tasks, each of length a_i, which may be processed in parallel.

We show that OS_k has a schedule of length T if and only if *PARTITION* has a solution. If *PARTITION* has a solution, we can deduce the following solution for OS_k (Figure 1):

- We schedule on M_1 the task O_{1k} of duration B, followed by the p relaxed jobs. This machine stops at time $3B$.
- We schedule on M_2 the relaxed jobs corresponding to the subset Y of the partition (total duration B), followed by the task O_{2k} (same duration) and the relaxed jobs of subset Z (same duration). The completion time of M_2 is then $3B$.
- On machine M_3, we schedule all relaxed jobs (total duration $2B$), followed by O_{3k} (duration B). M_3 is then free at time $3B$.

Conversely, if OS_k has a schedule of length $T = 3B$, then *PARTITION* has a solution. Indeed, as job k lasts $3B$, k is executed continuously in such a schedule. So there exists a machine M_i on which the task of k starts at time B. Such a situation is possible only if *PARTITION* has a solution.

This proves that problem OS_k is binary NP-hard even for three machines. □

Summarizing, we can get a new lower bound for the Open-Shop provided we can compute the optimal makespan for the relaxed problem OS_k. At first glance, the NP-hardness of OS_k compromises this eventuality. But, we show in the two subsequent sections that OS_k can be efficiently solved in practice.

3 Resolution of a special case OSS_k of OS_k

3.1 Definition of OSS_k and NP-hardness

We define OSS_k as the special case of OS_k in which the job k is processed through the machines in a fixed order. Assume without loss of generality that $M_1, M_2, ..., M_m$ is this fixed order. This problem is still NP-hard, since the reduction for problem OS_k presented in Proposition 2 is also applicable to problem OSS_k, for any order of the job k. However, we will show that it is pseudo-polynomially solvable via a decomposition into subset-sum problems.

We analyze OSS_k because the availability of a fast resolution method would allow OS_k to be solved by implicitly enumerating the processing routes of k, thus providing the desired bound for OS. In the sequel, we briefly present the principle of our algorithm for OSS_k, and clarify the process by a numerical example. Then, we can specify in details a pseudo-polynomial implementation and prove the optimality of the proposed algorithm.

3.2 Illustrative example of the solution method

OSS_k is solved in m steps. Roughly speaking, each step i considers a subproblem $OSS_k(i)$ whose the sole tasks are all tasks of job k, and the tasks of the first i machines on which job k is processed: $M_1, M_2, ..., M_i$. A set of schedules have been computed for $OSS_k(i-1)$ during the previous step. For each of these schedules, two schedules for $OSS_k(i)$ are determined in step i: in the first one, O_{ik} starts just after $O_{i-1,k}$; in the second, an idle time is allowed between these two tasks. This process is designed to obtain the optimal solution of OS_k among the solutions built in the last step.

We prefer to delay the proof of correctness (optimality) of the method and start with demonstrating some insight via a small numerical example. Consider the following matrix of processing times for a problem instance with four jobs and four machines.

	J_1	J_2	J_3	J_4	
M_1	309	354	10	307	980
M_2	254	357	197	145	953
M_3	201	100	319	334	954
M_4	236	172	464	114	986
	1000	983	990	900	

We assume a processing route (M_1, M_2, M_3, M_4). The classical bound LB (1000) is determined by job J_1. As our final goal is to obtain a bound at least as good as LB, we take this longest job as job k (see Proposition 1) and we solve problem OSS_1.

- In the first step, we solve the subproblem $OSS_1(1)$ consisting of the tasks of job J_1 and of set N_1. Recall that N_1 stands for the tasks to be executed on M_1, other than O_{1k}. The optimal solution of this problem is obvious (Figure 2): we schedule job J_1 at time 0, and we place contiguously after O_{11} the remaining tasks of machine M_1 (set N_1). This gives a solution S_1 with makespan 1000.

- In step 2, we solve the subproblem $OSS_1(2)$ whose input consists of the tasks of N_1 and N_2, and the ones of job J_1. We look back at solution S_1 from step 1 and consider two cases (Figure 3):

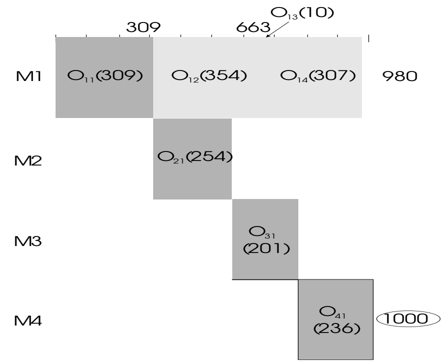

Figure 2: Optimal solution S_1 of step 1

- In the first case, we suppose that O_{21} starts just after O_{11} at time C_{11}, *i.e.* at the completion time of O_{11}. In this case, to minimize the completion time of machine M_2, we must fill as much as possible the gap before O_{21} on machine M_2. This problem is equivalent to finding a subset of tasks on M_2, whose sum of processing times is maximal but at most C_{11}. As the completion time of J_1 does not change, we obtain for this first case an optimal schedule S_{21} whose makespan 1065 equals the maximum of the makespan of S_1 and of the completion time of machine M_2.
- In the second case, we allow a non-empty pause in J_1 between O_{11} and O_{21}. To minimize the completion time of J_1, we must schedule on M_2, before O_{21}, a subset of tasks whose total duration is minimal but strictly greater than C_{11}. The completion time of M_2 is at most LB since this machine has no idle time. For this second case, we get a solution S_{22} with makespan 1033, which is equal to the maximum of the makespan of S_1 and of the completion time of J_1.

- In step 3, we take the tasks of N_1 through N_3, and the tasks of J_1 into account. We first construct two solutions based on the best solution S_{22} of the previous step (Figure 4).
  - O_{31} starts just after O_{21}: we schedule on M_3, before O_{31}, a subset of tasks filling as much as possible the gap of size C_{21}. The resulting schedule S_{31} ends at 1116, maximum of the makespan of S_{22} and of the completion time of M_3.
  - In the other case, O_{31} starts strictly after C_{21}. We schedule before O_{31} a subset of tasks minimizing the waiting time of J_1. This gives a solution S_{32} with makespan 1090, maximum of the makespan of S_{22} and of the completion time of J_1.

The process is repeated with solution S_{21}, to get two other solutions: S_{33} with

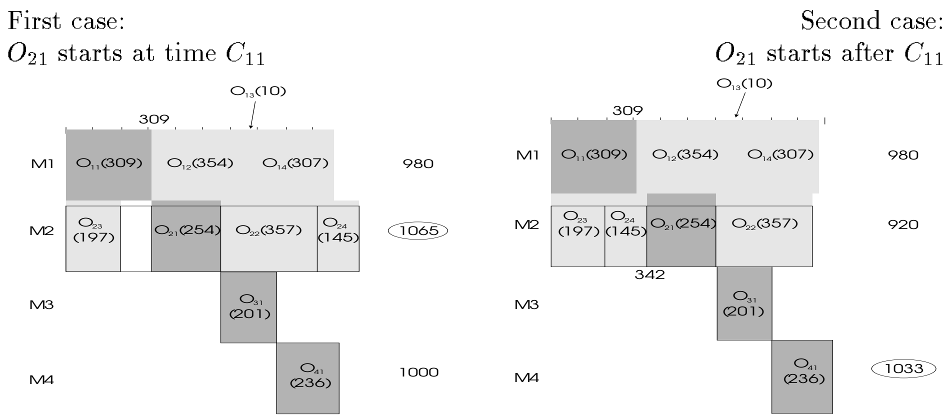

Figure 3: Solutions S_{21} and S_{22} of step 2, based on S_1 of step 1.

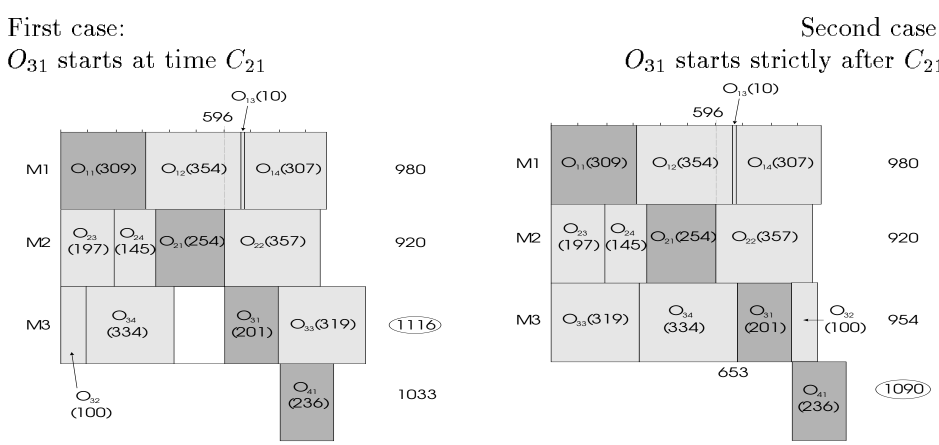

Figure 4: Solutions S_{31} and S_{32} based on S_{22} of step 2

First case: O_{31} starts at time C_{21}

Second case: O_{31} starts after C_{21}

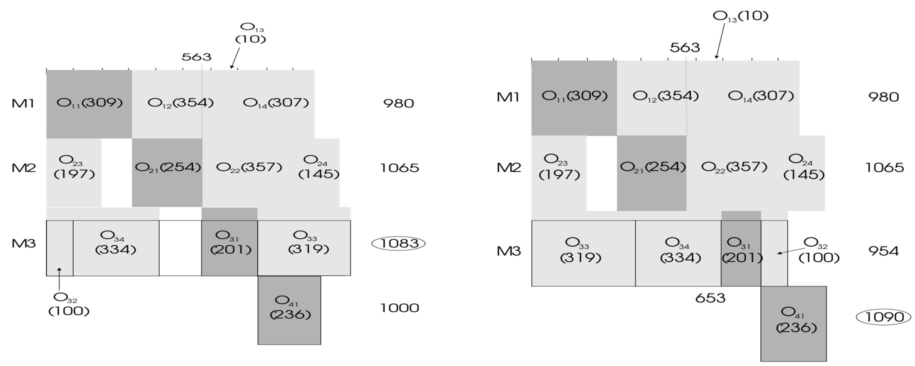

Figure 5: Solutions of step 3 based on S_{21} of step 2.

makespan 1083 and S_{34} with makespan 1090 (Figure 5). The best solution at step 3 is the one of smallest makespan among the four computed schedules *i.e.* S_{33} (1083).

- In step 4, the subproblem $OSS_1(4)$ is in fact OSS_1 itself since it includes the tasks of all machines.

  We first start from the best solution S_{33} of step 3 (Figure 6). We consider only the case in which O_{41} starts immediately after O_{31}, since we can schedule all tasks of N_4 before O_{41} in this solution.

  We obtain a solution of makespan 1083. No better schedules can be obtained from the other solutions of step 3 since they all end beyond 1083. We notice that 1083 is much better than the classical lower bound $LB = 1000$ we had at the beginning. Of course, this makespan concerns only the particular processing route (M_1, M_2, M_3, M_4) of J_1 : for solving OS_1 optimally, we should compute the minimum makespan for each possible sequence of operations and keep the *minimum* (and not the maximum), since the corresponding processing route of J_1 may appear in the optimal solutions of the initial Open-Shop OS.

3.3 Summary of the method

To summarize, in step 1, O_{1k} is scheduled at time 0. Then, at each step i and for each solution of step $i-1$, two solutions are constructed. In the first one, S', O_{ik} starts immediately after $O_{i-1,k}$ (Figure 7). In this case, we schedule before O_{ik} a subset of tasks whose total duration is maximal, but not greater than the completion time t of $O_{i-1,k}$. We call the total duration of the tasks (operations) of this subset $Down(i,t)$. The makespan of S' is then $C^{'}_{\max} := \max(C_{\max}, L_i + t - Down(i,t))$, where $C_{\max}$ denotes the makespan of the chosen solution from step $i-1$.

In the second solution S'' (Figure 8), we insert the shortest non-empty pause possible

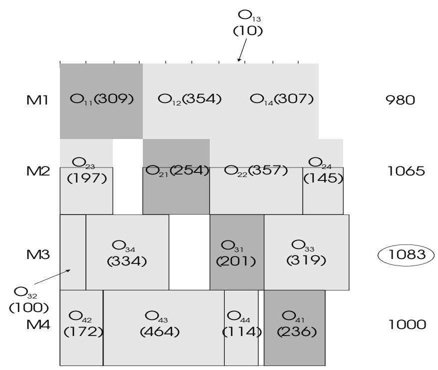

Figure 6: Optimal solution of OSS_k of step 4.

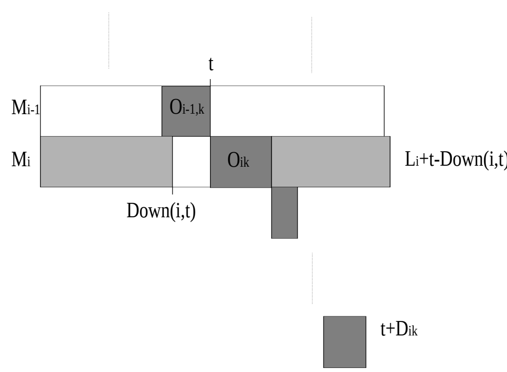

Figure 7: First solution S'

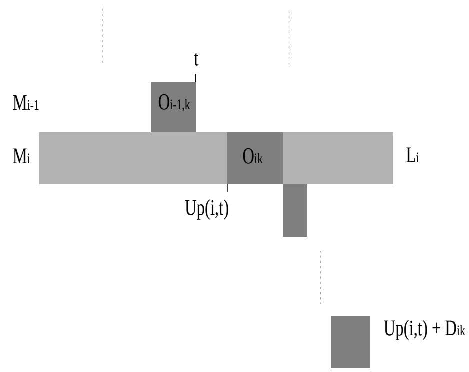

Figure 8: Second solution S''

between $O_{i-1,k}$ and O_{ik}, by scheduling before O_{ik} a subset of tasks whose total duration is minimal, but strictly greater than the completion time t of $O_{i-1,k}$. We call the weight of such a subset $Up(i,t)$. In this case, the makespan of S'' is $C''_{\max} := \max(C_{\max}, Up(i,t) + D_{ik})$.

These ideas raise three questions:

- How to compute efficiently the values of *Down* and *Up*?
- As described, the algorithm seems to build 2^{m-1} partial schedules. Is it possible to achieve a better complexity?
- Does the algorithm solve OSS_k optimally?

3.4 Computation of $Down(i,t)$ and $Up(i,t)$

For a subset Q of jobs and a machine M_i, let $x(i,Q) = \sum_{j \in Q} p_{ij}$. For a given integer t, this notation allows a concise definition of $Down(i,t)$ and $Up(i,t)$:

$$Down(i,t) = max_{Q \subset J \backslash \{k\}} \{x(i,Q) \mid x(i,Q) \leq t\}$$

$$Up(i,t) = min_{Q \subset J \backslash \{k\}} \{x(i,Q) \mid x(i,Q) > t\}$$

We recognize in these expressions classical *subset-sum* (or *stick-stacking*) *problems*. They are NP-hard but can efficiently be solved in practice by a dynamic programming (DP) method [5]. When adapted to our problem OSS_k and one machine M_i, the method enumerates all possible partial sums of the processing times of set N_i. In other words, we obtain all values of $x(i,Q)$ which can be achieved by all subsets $Q \subset J \backslash \{k\}$, regardless of t. This is done in n iterations, the iteration j computing all values of $x(i,Q)$ achievable by subsets $Q \subset \{1, \ldots, j\} \backslash \{k\}$.

```
Initialize array Possible to False
Last := 0
Possible[Last] := True
For j:=1 to n, j ≠ k
    For p:=Last downto 0
        If Possible[p]=True then
          q := p + p_ij
          Possible[q] := True
          If (q > Last) and (q ≤ LB) then
            Last := q
          EndIf
        EndIf
    EndFor
EndFor
```

Figure 9: Algorithm DP for all subsets of tasks of machine M_i

The algorithm given in Figure 9 is a direct translation of this process. The array *Possible* contains $LB + 1$ booleans indexed from 0 onwards. Its element number q is equal to $True$ iff there exists a value $x(i, Q) = q$. The size $LB + 1$ is sufficient, since the load of machine M_i is at most LB by definition of this bound. The variable *Last* gives the maximum value reached so far, it allows to scan only the useful parts of the arrays.

From the two nested loops, it is clear that the algorithm DP runs in $O(n \times LB)$. As it yields all possible values of $x(i, Q)$, it can be run once for each machine at the beginning of the resolution of OSS_k. Later, for any given t, we just check $Possible[t]$. $Down(i, t)$ is the greatest index $u \leq t$ such as $Possible[u] = True$, and $Up(i, t)$ is the smallest index $v > t$ such as $Possible[v] = True$. Note that these values can be found in $O(1)$ if the $True$ elements of array *Possible* are linked forwards and backwards by pointers. These links can be calculated at the end of algorithm DP without changing its complexity.

For instance, consider the numerical example of the previous section, in which the tasks taken into account on M_2 have durations 357, 197 and 145 (the task of the non-relaxed job J_1 being excluded). The DP algorithm sets to $True$ the elements of *Possible* with the following subscripts, sorted in increasing order:

145	197	342	357	502	554	699

Suppose task O_{1k} is scheduled at time 0 on M_1 and thus ends at 309. Using the method previously described, we find $Down(2, 309) = 197$ and $Up(2, 309) = 342$.

3.5 Pseudo-polynomial algorithm for OSS_k

We now have all the elements to specify a pseudo-polynomial algorithm A for solving OSS_k. The non-relaxed job k is given, and is processed without loss of generality on the machines M_1 to M_m in this order.

To speed up the computations and discard many schedules, we first apply a simple heuristic H to get an upper bound UB for OSS_k. This heuristic is a fast and greedy version of the process summarized in subsection 3.3. In step 1, it starts from the same trivial schedule. In step 2 it only keeps for step 3 the schedule of minimum makespan among S' and S'', and so on. As it considers only one schedule in each step, this heuristic runs in $O(m)$. The following proposition states an interesting property of H which will be useful to establish the pseudo-polynomial complexity of A.

Proposition 3
The heuristic H constructs schedules of makespan $UB \leq 2 \cdot LB$.

Proof
We consider 2 cases:

- Job k ends before UB. The makespan is then due to the completion time $C_i \geq L_i$ of a machine M_i $(i < m)$. The largest value of C_i is achieved by putting on M_i, before O_{ik}, the largest possible gap. This gap lasts at most $L_i - p_{ik}$. Hence, $C_i \leq 2.L_i \leq 2 \cdot LB$.
- Job k ends at UB. If it is processed in no-wait, then $UB = D_k = LB$. If k has a pause between, for instance, M_{i-1} and M_i $(i > 1)$, then $C_{ik} = Up(i, C_{i-1,k}) + p_{ik} \leq L_i \leq LB$, since Up corresponds to a subset of tasks of M_i. This applies in particular to the last pause of k. Thus, k ends at $C_{ik} + D_{i+1,k} \leq LB + LB \leq 2 \cdot LB$.

□

The algorithm A for solving OSS_k is given in Figure 10.

At iteration i, the algorithm computes the schedules for all tasks of job k and all tasks of machines M_1 to M_i (what we call subproblem $OSS_k(i)$). For the first iteration, there is only one schedule of makespan LB and with O_{1k} starting at time 0.

In the illustrative example of section 3.2, we have used an inefficient process which doubles the number of schedules scanned in each iteration. In fact, at the end of iteration i, the actual algorithm only keeps schedules with different C_{ik} values. If two schedules S_1 and S_2 have the same value for C_{ik}, we keep the one with smallest makespan. The reason is simple : in the two families of schedules generated from S_1 and S_2 in subsequent iterations, the machines M_{i+1} to M_m have identical completion times. Moreover, the only information stored for each schedule is a pair $(C_{ik}, C_{\max})$: this is sufficient, since the goal is to compute the optimal makespan of OSS_k, but not the exact composition of an optimal schedule. These details are crucial to achieve a pseudo-polynomial complexity.

The implementation stores the schedules at the beginning of iteration i in an array $ZOld$ of $UB + 1$ elements indexed from 0. If a schedule with $C_{i-1,k} = t$ and makespan C was built at the previous iteration, then $ZOld[t] = C$, else $ZOld[t]$ is set to a huge number. Iteration i builds the schedules for $OSS_k[i]$ in a second array $ZNew$ which overwrites $ZOld$ for the next iteration.

Proposition 4
Algorithm A has a pseudo-polynomial complexity of $O(mn \times LB)$.

```
Apply algorithm DP to each machine and store the results
UB := the makespan computed by the greedy heuristic H for OSS_k
If UB = LB Stop
Create two arrays ZOld and ZNew of UB + 1 integers
Initialize ZOld to a large value Huge
ZOld[p_1k] := LB (* This defines the trivial schedule of step 1 *)
For i:=2 to m
    Initialize ZNew to Huge
    For t = 0 to UB if ZOld[t] ≠ Huge
        (* Here we have a schedule with C_{i-1,k} = t and C_max = ZOld[t] *)
        C_max = ZOld[t]
        Compute Down(i,t) and Up(i,t) from the results of DP
        (* First solution S': M_i is delayed, but not k *)
        C_ik := t + p_ik (* no pause between O_ik and O_{i-1,k} *)
        C'_max = max(C_max, L_i + t - Down(i,t))
        If C'_max < UB and C'_max < ZNew[C_ik] then ZNew[C_ik] = C'_max
        (* Second solution S'': k is delayed, but not M_i *)
        C_ik := Up(i,t) + p_ik (* minimum gap between O_{i-1,k} and O_ik *)
        C''_max = max(C_max, Up(i,t) + D_ik)
        If C''_max < UB and C''_max < ZNew[C_ik], then ZNew[C_ik] = C''_max
    EndFor
    ZOld = ZNew
EndFor
C*_max(OSS_k) := minimum value ZOld
```

Figure 10: Pseudo-polynomial time algorithm A for problem OSS_k

Proof
At the beginning, for each machine, we apply the $O(n \cdot LB)$ DP algorithm once to store the durations which can be achieved by all subsets of tasks. This costs $O(mn \cdot LB)$. The greedy heuristic H is also executed to compute UB, this costs $O(m)$.

Then, we have a main loop with $m-1$ iterations. In each iteration i, we scan the array $ZOld$ which contains $UB+1$ elements. As explained in section 3.4, $Down$ and Up can be computed in $O(1)$ for each schedule S in $ZOld$, provided the results of DP are stored in a proper way. The complexity of the main loop is then $O(m \cdot UB)$, and the overall complexity is $O(m \cdot UB + mn \cdot LB) = O(m \cdot (UB + n \cdot LB))$. But, since the initial heuristic solution is computed by a truncated version of A, we have $UB \leq 2 \cdot LB$ from proposition 3. LB can then replace UB in the previous complexity, which becomes $O(mn \times LB)$. □

3.6 Optimality proof

We prove that algorithm A finds the optimal solution of OSS_k. Recall a definition and a property (taken from French [2]):

- A schedule S is *active* if a) it is feasible and b) no task can be started earlier without either delaying some other task or violating the feasibility.
- In any shop problem, to minimize a regular measure of performance, it is sufficient to consider active schedules.

The optimality is proved in two steps: we first show that A generates a subset of active schedules, and then that it generates at least one optimal schedule. Consider any instance of OSS_k, and let $S(A)$ be the set of solutions built with algorithm A, and ACT the set of active schedules for this instance.

Proposition 5
$S(A) \subseteq ACT$.

Proof
If there is a pause on a machine M_i, this pause is unique and occurs before O_{ik}. By definition of $Down$, no task processed after O_{ik} can be started during this pause without delaying O_{ik}. Moreover, O_{ik} cannot start earlier without overlapping with $O_{i-1,k}$. Therefore, any schedule generated by A is active. □

Proposition 6
$S(A)$ contains at least one optimal schedule

Proof
We just give the idea and leave the details to the reader. Any active schedule S for OSS_k can be transformed into an active schedule belonging to $S(A)$, without increasing makespan : to achieve that, we run algorithm A and we check at each step i (on each machine M_i) if A finds the same completion time as in S for M_i. Let $t = C_{i-1,k}$. A discrepancy means that the subset of tasks before O_{ik} in S has a total duration different from $Down(i,t)$ and $Up(i,t)$, and this can be corrected easily. The final schedule adjusted by this process belongs to $S(A)$. □

4 Resolution of OS_k to obtain the new lower bound

We have shown in the previous section how to solve problem OSS_k in pseudo-polynomial time. Recall that OSS_k is a special case of problem OS_k in which the sequence of operations of the non-relaxed job k is predefined. Our main goal is to solve OS_k, since its optimal makespan $C^*_{\max}(OS_k)$ is a lower bound for the initial problem OS. As hundreds of problems OSS_k can be solved per second, even on a small PC, we tried to apply algorithm A within an implicit enumeration of all possible processing routes of k. After several computational experiments, we adopted the following branch-and-bound structure, in which algorithm A is in fact executed incrementally along the branches of the search tree.

4.1 Definition of nodes

A node at level p in the search tree contains a list of the p first machines on which k is processed, and one of the solutions of type S' or S'' found for this processing route by step p of algorithm A. The root node, at level 0, is a dummy node in which all machines are free.

4.2 Branching rule

At level p, $p > 1$, we build $2(m - p + 1)$ child nodes from a parent node. In the parent node of level $p-1$, we have one schedule S for subproblem $OSS_k(p-1)$. For the $p-1$ first machines of k defined in the parent node, this subproblem consists of the tasks of these machines, and the tasks of k. To generate one child, we first select the p-th machine on which k continues its execution: there remains $m - p + 1$ free machines at level p. Then we compute either the solution S' or S'' as in algorithm A, according to the two possible positions of the task of k on the p-th machine.

4.3 Lower bound to prune the search tree

We use the greedy heuristic of algorithm A to get a first upper bound UB on the makespan at the root node. During the search, we update UB each time a better provisional solution is found. When constructing a node at level p, we compute the makespan $C'_{\max}$ or $C''_{\max}$ of the partial schedule associated with that node. This gives a lower bound for all schedules which can be obtained by continuing the execution of job k beyond the p-th machine. If this lower bound is not smaller than UB, the node is pruned.

4.4 Search strategy

After some preliminary testing, we took as search strategy a kind of hybrid between the classical depth-first and frontier searches. We proceed like in a pure depth-first search but, instead of developing the first child node, we generate and evaluate all children nodes and we branch on the child of smallest lower bound. The pure depth-first and frontier searches are less efficient: we found that the average number of nodes they generate is 1.5 to 2 times larger than for the hybrid method.

5 Computational evaluation of the new lower bound

5.1 Implementation of algorithms

The branch-and-bound method which computes the new lower bound has been implemented in Borland Pascal 7.0 (compatible with the Object Pascal 9.0 which replaces it in Borland Delphi) and tested on a Pentium PC clocked at 166 MHz under the operating system Windows 95. In all our tests, we considered square ($m = n$) Open-Shop problems because they are known to be more difficult than rectangular problems, as in the Job-Shop problem.

5.2 Hard instances from the literature

On Taillard's problems, the new bound is equal to LB. This is not surprising, since we saw in Section 1 that practically all these problems have been optimally solved, and that the optimum is always equal to LB, except for very small instances.

The situation becomes more interesting on the hard problems designed by Brucker *et al.*. In these problems, there is at least one line sum equal to 1000, and the classical lower bound LB is equal to this load. All job durations and machine loads are either equal to

Table 1
Hurink's hard problems with $LB = 1000$

Name	Opt	NB on J_1	Nodes	CPU (s)	NB all jobs	Nodes	CPU (s)
joh3x3-1	1127	1047	0	0.0000	1047	4	0.0000
joh3x3-2	1084	1012	2	0.0065	1012	4	0.0065
joh4x4	1055	1043	8	0.0065	1043	18	0.0054
joh4x4-1	1180	1001	5	0.0065	1083	16	0.0065
joh4x4-2	1071	1020	6	0.0055	1035	22	0.0065
joh5x5	1042	1007	15	0.0065	1008	63	0.0122
joh5x5-1	1054	1011	14	0.0065	1025	63	0.0122
joh5x5-2	1063	1009	18	0.0065	1011	74	0.0122
joh6x6	1056	1004	17	0.0054	1004	96	0.0243
joh6x6-1	1045	1006	28	0.0065	1006	106	0.0244
joh6x6-2	1063	1003	23	0.0055	1004	132	0.0177
joh7x7	-	1001	54	0.0122	1002	305	0.0488
joh7x7-1	1055	1002	25	0.0054	1004	210	0.0366
joh7x7-2	1056	1002	37	0.0066	1004	217	0.0366
joh8x8-1	-	1001	44	0.0122	1002	442	0.0676
joh8x8-2	-	1001	47	0.0054	1002	329	0.0555

1000 or very close to this value. One can imagine the hardness of such instances: the least idle time put in a schedule by any algorithm delays the makespan beyond 1000.

Table 1 gives the results for the 16 hardest Hurink's instances, in which all line sums are exactly equal to 1000. NB denotes the new bound. Note that the branch-and-bound method from Brucker *et al.* [1], which is currently the best exact method for the Open-Shop, fails to solve three problems of this set, including one 7×7 problem. For each problem we mention the value of NB when taking J_1 as job k, and the best bound obtained by considering each job in turn as job k. Our new bound improves LB in all cases, although the computation time increases compared to uniform problems. This indicates that the hardness of the relaxed Open-Shop used for the new bound seems to increase with the difficulty of the initial Open-Shop.

5.3 Harder instances from a specially designed generator

Last but not least, we developed our own random generator to produce even harder instances. Its details are given by Algorithm G presented in Figure 11. G needs m and n as input. It uses three parameters K, NP and FP, the role of which is explained in the sequel. The first main loop builds a matrix P with all line sums equal to K. All elements are equal to K div m (div being the euclidean division provided by Pascal), except on the diagonal where the remainder K mod m is added.

The second main loop performs NP perturbations. Each perturbation randomly selects two elements p_{ij} and p_{kl} with $i \neq k$ and $j \neq l$, from which a certain amount of work will be removed. The maximum amount of removable work is the minimum of the two durations, minus 1 to avoid creating tasks of length 0. A fixed part of this maximum

```
For i:=1 to m
    For j:=1 to n
        p_ij := K div m
    EndFor
    p_ii := p_ii + (K mod m)
Endfor
For Perturb := 1 to NP
    Select randomly 2 tasks p_ij and p_kl with i ≠ k and j ≠ l
    Movable := min(p_ij, p_kl) − 1
    Mandatory := Trunc(Movable * FP)
    Moved := Mandatory + a random integer in [0, Movable - Mandatory]
    Subtract Moved from p_ij and p_kl
    Add Moved to p_il and p_kj
EndFor
```

Figure 11: Algorithm G for the random generation of hard instances

Table 2
Problems from our generator, with $LB = 1000$ (NB computed on all jobs)

Size	FP	Number of problems with $NB > LB$	Average NB/LB	Average NB/LB with $NB > LB$	NB max	Average number of nodes	Average CPU time
3x3	0.00	979	1031.19	1031.86	1150	1.07	0.004
4x4	0.30	1000	1040.37	1040.37	1246	4.65	0.010
5x5	0.90	1000	1041.13	1041.13	1206	10.84	0.030
6x6	0.95	1000	1036.16	1036.16	1264	28.70	0.088
7x7	0.95	1000	1025.45	1025.45	1141	99.87	0.288
8x8	0.95	1000	1014.26	1014.26	1142	310.58	0.8017
9x9	0.95	988	1007.44	1007.53	1091	786.50	2.293
10x10	0.95	610	1002.33	1003.82	1065	1523.24	3.916

is really removed, its amount being defined by the FP (*fixed percentage*) parameter. An additional number of time units is randomly drawn in the remaining amount and is also removed. Tasks p_{il} and p_{kj} receive the amount removed from the two first tasks, to keep all line sums equal to K.

In the evaluations with this generator, we took $K = 1000$ as in Hurink's problems and $NP = n^2m$ perturbations. More perturbations increase the new bound only slightly, at the expense of a much greater generation time. We made runs of 1000 instances for each size. It appeared that the hardness (measured as the average value of the bound) is maximized for a precise value of FP which depends on the size. Starting from small instances, the best value of FP increases rapidly to 0.95. We give in Table 2 the results obtained for these best values of FP.

Our generator G produces very hard instances. The computation time of the bound increases a lot, but remains quite reasonable: note that there exist Open-Shop problems with 7 jobs and 7 machines which cannot be solved in 50 hours on a SUN Sparc-5 workstation by the existing branch-and-bound methods.

We were impressed by the maximum bound found in each series of 1000 instances.

Table 3
Overview of the hardest instances found by our generator

Size	NB Min	NB Max
3x3	1156	1164
4x4	1229	1279
5x5	1232	1287
6x6	1229	1299
7x7	1150	1270
8x8	1109	1218
9x9	1078	1180
10x10	1044	1075

We decided to run overnight our benchmarking program on very long series of problems (20,000 to 150,000 depending on the size). We found for each size a set of 20 extremely hard instances, with NB/LB ratios of around 1.3 on problem sizes 5×5 to 7×7. The minimum and maximum value of the bounds in each set are given in Table 3.

Here is for instance the hardest 6×6 instance found, with $NB = 1299$.

	J_1	J_2	J_3	J_4	J_5	J_6
M_1	1	804	156	37	1	1
M_2	644	11	15	8	6	316
M_3	1	173	184	493	148	1
M_4	1	3	643	10	1	342
M_5	1	8	1	1	650	339
M_6	352	1	1	451	194	1

6 Conclusion and further research

In this paper, we propose for the Open-Shop problem the first lower bound on the makespan which improves the classical bound LB. The computation of this bound is theoretically NP-Hard, but can quickly be performed in practice. This is possible because the subproblems in which the processing route of k is imposed are equivalent to subset-sum problems for which an efficient DP algorithm is available.

Since the NP-hard relaxed problem we solve for our lower bound is still an Open-Shop problem, our work also refines the frontier between polynomial and NP-hard special cases of the Open-Shop.

The computational evaluation showed that the new bound outperforms the classical bound LB on hard instances. We designed a special random generator to perform this evaluation. As a by-product, we obtained a set of very hard instances that we propose to the scheduling research community. These instances resist to the best branch-and-bound methods available as from size 7×7. This shows that, contrary to a widespread opinion, the Open-Shop may be as difficult as the Flow-Shop and Job-Shop.

We are currently working on the application of this bound in a branch-and-bound algorithm for the Open-Shop. As the majority of *B&B* algorithms for shop problems branch by fixing disjunctions, the major difficulty seems to generalize our bound to handle precedence constraints. Up to now, we have preliminary results only: when the bound is added to the algorithm of Brucker *et al.* with some other refinements, the number of nodes decreases, sometimes by a factor 10. For example, we tried to solve the 6×6 open-shop given in paragraph 5.3 with the Branch-and-Bound of [1]. In this example, the optimal value is equal to our new lower bound (1299). Without this new bound, the optimal solution is found but its optimality is not proved after the exploration of 300000 nodes (more than 10 hours of CPU times on a Pentium PC clocked at 133 MHz). If our new lower bound is used, the exploration stops after 210 nodes only. In our opinion, the investigation of specific and totally new branching schemes is necessary for the Open-Shop, if one wants to solve any 10×10 instance as this is the case already for the Flow-Shop and the Job-Shop.

7 Appendix

The problems designed by Taillard are described in [7], but they can be obtained on the web, from the famous OR-Library maintained by J.E. Beasley :

http://mscmga.ms.ic.ac.uk/info.html.

An FTP site is also available for Hurink's instances:

ftp.mathematik.uni-osnabrueck.de in the path */pub/osm/preprints/software/openshop.*

The Pascal code of our generator and the seeds required to build the hardest instances summarized in Table 3 can be requested by mail or e-mail.

References

[1] P. Brucker, J. Hurink, B. Jurisch and B. Wöstmann, *A Branch-and-Bound Algorithm for the Open-Shop Problem*, Discrete Applied Mathematics, 76, 43–59, 1997.

[2] S. French, *Sequencing and Scheduling: an introduction to the Mathematics of the Job-Shop*, Ellis Horwood, 1990.

[3] T. Gonzales and S. Sahni, *Open Shop Scheduling To Minimize Finish Time*, Journal of the ACM, 23(4), 665–679, 1976.

[4] C. Guéret and C. Prins, *Classical and New Heuristics for the Open-Shop Problem*, European Journal of Operational Research, 107(2), 306–314, 1998.

[5] S. Martello and P. Toth, *Knapsack problems*, Wiley, 1990.

[6] I.G. Drobouchevitch, V.A. Strusevich, *A polynomial algorithm for the three-machine open-shop with a bottleneck machine*, CASSM R&D paper 13, University of Greenwich, United Kingdom, August 1997.

[7] E. Taillard, *Benchmarks for basic scheduling problems*, Research Report ORWP 89/21, University of Lausanne, Switzerland, 1989.

[8] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov and D.B. Shmoys, *Short shop schedules*, Operations Research, 45, 288-294, 1997.