A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles

L. DEROUSSI*†, M. GOURGAND‡ and N. TCHERNEV§

†IUT de Montluçon, France
‡Campus des Cézeaux, France
§IUP Gestion et Management des Entreprises, France

(Revision received May 2006)

In this paper we address the problem of simultaneous scheduling of machines and vehicles in flexible manufacturing systems. The studied problem is a job shop where the jobs have to be transported between the machines by automatic guided vehicles. In addition to the processing of jobs, we consider the transportation aspect as an integral part of the optimization process. To deal with this problem, we propose a new solution representation based on vehicles rather than machines. Each solution can thus be evaluated using a discrete event approach. An efficient neighbouring system is then described and implemented into three different metaheuristics: iterated local search, simulated annealing and their hybridisation. Computational results are presented for a benchmark of 40 literature instances. New upper bounds are found for 11 of them, showing the effectiveness of the presented approach.

Keywords: Job-shop problem; Metaheuristics; Neighbouring systems; Flexible manufacturing systems; Automated guided vehicles

1. Introduction

A Flexible Manufacturing System (FMS) can be described as follows: FMS consists of several machine tools along with part and tool handling devices such as robots, arranged so that it can handle any family of parts for which it has been designed and developed. These systems require a high investment but they have a high potential. As a result, scheduling of FMS has been extensively investigated during the last three decades.

This paper is devoted to the study of the classical job shop problem where transportation times are considered. It is classified as a $JR|t_{kl}, t'_{kl}|C_{\text{max}}$ problem according to the $\alpha|\beta|\gamma$-notation introduced by Graham et al. (1979) and extended by Knust (1999) for transportation problems. $J$ indicates a job shop, $R$ indicates that we have a limited number of identical vehicles (robots) and all jobs can be transported by any of the robots. $t_{kl}$ indicates that we have job-independent, but machine-dependent loaded travel times. $t'_{kl}$ indicates that we have machine-dependent empty travel times. The objective function to minimize is the makespan $C_{\text{max}}$.

*Corresponding author. Email: deroussi@moniut.univ-bpclermont.fr
Several authors note the necessity to carefully plan the interactions between process and transportation activities (Raman et al. 1986, Ülusoy and Bilge 1993, Knust 1999). Unfortunately, the resulting problem is quite difficult to solve. We have to deal with two NP-hard problems simultaneously: the job-shop scheduling problem (Lenstra and Rinnooy Kan 1979) and the vehicle scheduling problem, which is similar to a pick up and delivery problem (Lenstra and Rinnooy Kan 1981). Additionally, problems with transportation times are NP-hard since they are a generalization of classical NP-hard shop problems. More precisely, concerning the job shop, Knust (1999) writes: ‘The classical job shop problems J2∥C_{\text{max}} and J3|pq=1|C_{\text{max}} are both strongly NP-hard, which implies that only very restrictive special cases with transportation times may be polynomially solved’.

As a consequence, exact methods are mainly used for the study of simple or particular FMS, with strict assumptions. Thus, Blazewicz et al. (1991) study a FMS in which identical parallel machines are laid out in loop. Raman et al. (1986) present a MIP (Mixed Integer Programming) formulation of this problem but with the unrealistic assumption that the vehicles come back to the load/unload (LU) station after each achieved transport. Bilge and Ülusoy (1995) propose a MIP formulation raising this constraint on the vehicles. According to the authors, the resulting model is intractable in practice, because of its nonlinearity and its size.

Approximated methods are well adapted to study most of the FMS, but within an offline framework. Nevertheless, many works are dedicated to simplified forms of this problem. There are essentially two kinds of simplifications. The first ones consist in defining dispatching rules for the vehicles, for example described and studied by Egbelu and Tanchoco (1984) and Le Anh and De Koster (2005). These rules allow to determine how AGVs move for a given scheduling of machines. The second ones are the restriction of the material handling system at only one AGV. As an illustration, Soylu et al. (2000) and Hurink and Knust (2002) propose, respectively, neural networks and tabu search approaches. Finally, few works are undertaken on the job shop with transportation. Ülusoy and Bilge (1993) and Bilge and Ülusoy (1995) propose an iterative method based on the decomposition of the master problem into two sub-problems. Ülusoy et al. (1997) propose a genetic algorithm for this problem. Abdelmaguid et al. (2004) describe a hybrid method composed of a genetic algorithm for the scheduling of machines and a heuristic for the scheduling of vehicles.

For online scheduling of such systems, simulation is often the only available tool. Discrete event simulation is used for various purposes: for example to improve the AGV design process (Gobal and Kasilingam 1991), to compare the vehicles’ online dispatching rules (Egbelu and Tanchoco 1984) or to test the problems of collision between the vehicles (Revieiotis 2000).

Nuhut (1999), Qiu et al. (2002) and Le Anh and De Koster (2006) present more complete surveys of the job-shop problem with transportation.

For this problem, most of the (small) instances considered in this paper remain unsolved today. Ganesharajah et al. (1998) note that due to the intractability of the joint scheduling problem, the literature contains mainly heuristic approaches. In this paper, we propose a new effective metaheuristic approach for the offline JR|t_{kl}, t'_{kl}|C_{\text{max}} problem.

This paper is articulated as follows. In section 2 we present a problem formulation and we propose a new solution representation based on vehicles.
In section 3 we show how feasible solutions can be evaluated using a discrete event approach. Section 4 is devoted to the description of an efficient neighboring system. The metaheuristics implemented are described in section 5 and some computational results can be found in section 6.

2. The job-shop problem with transportation: formulation and solution representation

In this section we introduce the job-shop problem with transportation using the Brucker’s notations (2001), we present the chosen solution representation and we discuss its advantages and its drawbacks.

2.1 Problem formulation

In a job-shop scheduling problem, we have a set $J = \{J_1, \ldots, J_n\}$ of $n$ independent jobs that have to be processed without pre-emption on a set $M = \{M_0, M_1, \ldots, M_m\}$ of $m+1$ machines ($M_0$ represents the load/unload or LU station from which jobs enter and leave the system). Each job $J_i \in J$ consists of a sequence of $n_i$ operations $o_{ij}$. Let us note $O_i = \{o_{ij}, j = 1, \ldots, n_i\}$ the set of operations of job $J_i$, and $O = \bigcup_{i=1}^{n} O_i$ the set of $o = \sum_{i=1}^{n} n_i$ operations. There is a machine $\mu_{ij} \in \{M_0, \ldots, M_m\}$ and a processing time $p_{ij}$ associated with each operation $o_{ij}$. Additionally, a vehicle has to transport a job whenever it changes from one machine to another. We have a given set $V = \{V_1, \ldots, V_k\}$ of $k$ vehicles. We assume that transportation times are only machine-dependant. $t(M_i, M_j)$ and $t'(M_i, M_j)$ indicate, respectively, the loaded travel time and the empty travel time from machine $M_i$ to machine $M_j$ ($i, j = 0, \ldots, m$). Vehicles can handle at most one job at a time.

The objective function is the minimizing time required to complete all jobs or makespan.

2.2 Representation of the search space

It is well known that the efficiency of metaheuristics highly depends on finding a suitable representation of the search space. For that a new solution representation is introduced, based on vehicles rather than on machines. Each operation $o_{ij} \in O$ requires a transportation task we will note $e_{o_{ij}}$, and we propose to schedule the set $\tilde{O}$ of transportation tasks on the material handling system. In the following sections of this paper, we will designate by operations the members of $O$, and by transportation tasks (or tasks) the members of $\tilde{O}$. Let us give in more detail the link between $O$ and $\tilde{O}$.

Each operation $o_{ij}$ requires the transportation of job $J_i$ from the previous machine that is defined in its sequence of operations ($M_0$ if $j = 1$ or $\mu_{i,j-1}$ if $j > 1$) to the machine $\mu_{ij}$. So a transportation task $\tilde{o}_{ij}$ is associated with each operation $o_{ij}$. Moreover, when the processing of a job is completed, a vehicle has to transport it to the unload station $M_0$. So we consider an additional task $\tilde{o}_{in_i+1}$ for each job. To find again a link between operations and tasks, a fictitious operation $o_{in_i+1}$ is added to the sequence of each job. Obviously, we have $\mu_{in_i+1} = M_0$ and $p_{in_i+1} = 0$. We give
in Table 1 an example of a feasible representation for jobset 1 (see the appendix section for the jobsets definition).

We will indicate by $v(\bar{e}_{ij}) \in V$ the vehicle that completes the task $\bar{e}_{ij}$. $\lambda(\bar{e}_{ij})$ represents the position of the task $\bar{e}_{ij}$ in the schedule. For example, in Table 1 we have $v(\bar{e}_{13}) = V_2$ and $\lambda(\bar{e}_{13}) = 3$.

### 2.3 Advantages of the proposed solution representation

The proposed representation has major differences with those of the literature. (Ulusoy et al. 1997) represent a schedule by a chromosome in which each gene has two data: the operation to be completed and the vehicle at which this operation is assigned. The solution given in Table 1 is represented by the chromosome $[(\sigma_{11}, V_1), (\sigma_{21}, V_2), (\sigma_{41}, V_3), \ldots, (\sigma_{52}, V_1), (\sigma_{34}, V_3), (\sigma_{53}, V_3)]$. The tasks assigned to one vehicle (for example $\sigma_{11}, \sigma_{31}, \sigma_{22}, \sigma_{14}, \sigma_{23}$, and $\sigma_{52}$ for vehicle 1) are transported according to the order given by the chromosome. However, the same solution can also be represented by the chromosome $[(\sigma_{11}, V_1), \ldots, (\sigma_{52}, V_1), (\sigma_{21}, V_2), \ldots, (\sigma_{33}, V_2), (\sigma_{41}, V_3), \ldots, (\sigma_{53}, V_3)]$, and by many others. As a consequence, one schedule can be represented by many chromosomes, which define an equivalence class. Our representation consists in working with the set of equivalence class representatives. The immediate consequence is that the search space is significantly reduced, and thus probably easier to explore.

Abdelmaguid et al. (2004) get around this difficulty by using a chromosome in which only the order of the operations is mentioned. Thus, the chromosome $[\sigma_{11}, \sigma_{21}, \sigma_{41}, \ldots, \sigma_{52}, \sigma_{34}, \sigma_{53}]$ indicates the order in which the operations must be treated. A heuristic assigns successively each task to the ‘best’ vehicle (the chosen criterion is the vehicle that will carry out this task as soon as possible). The authors came back thus to the classical job-shop problem $J||C_{max}$, integrating dynamically the transportation times into the makespan evaluation. The advantage is that the number of solutions is considerably reduced. Nevertheless, we can see that vehicle assignments are locally optimal, but not necessarily globally.

The use of metaheuristics implies that many solutions must be considered. Such approach requires a makespan evaluation with a computationally fast evaluation subroutine. We present it in the next section.

### 3. Solutions evaluation

The makespan evaluation is described in algorithm 1 using a discrete event approach. $T(\sigma_j)$ (respectively $T(\bar{e}_{ij})$) indicates the completion time of the operation $\sigma_j$.

| Vehicle | Task schedule |
|---------|---------------|
| AGV1   | $\bar{e}_{11}$ $\bar{e}_{31}$ $\bar{e}_{22}$ $\bar{e}_{14}$ $\bar{e}_{23}$ $\bar{e}_{52}$ |
| AGV2   | $\bar{e}_{21}$ $\bar{e}_{42}$ $\bar{e}_{13}$ $\bar{e}_{32}$ $\bar{e}_{33}$ |
| AGV3   | $\bar{e}_{41}$ $\bar{e}_{12}$ $\bar{e}_{51}$ $\bar{e}_{43}$ $\bar{e}_{24}$ $\bar{e}_{34}$ $\bar{e}_{53}$ |
(respectively the transportation task $\tilde{\sigma}_{ij}$). We use the notion of available task, which can be defined as follows:

A transportation task is said to be available at a given time $t$ if and only if it has not been carried out yet and it satisfies one of the two following conditions:

- It is associated to the first operation of the job sequence;
- the previous operation (if it exists) in the job sequence is completed at time $t$.

The available tasks correspond in fact to the transports that can be effectively accomplished at a given moment.

We can note the following remarks about algorithm 1:

(a) At the initial state, the only available tasks $\tilde{\sigma}_{ij}$ are those for which $j=1$ (corresponding to the first operation defined in the sequence of the job $J_i$).
(b) A new event is created if the following transportation task of the vehicle $V_l$ is available (its previous corresponding operation has been already completed).
(c) The task $\tilde{\sigma}_{ij}$ has been just completed. This means that the task $\tilde{\sigma}_{ij+1}$ (if it exists) becomes an available one. A vehicle waiting for this task can now accomplish it, and an event is created. Note that if tasks $\tilde{\sigma}_{ij}$ and $\tilde{\sigma}_{ij+1}$ are carried out by the same vehicle, this event has been already created in case (b).

Algorithm 1: Makespan evaluation

// Initialization
For each task $\tilde{\sigma}_{ij} \in \tilde{O}$ Do

\[ T(\tilde{\sigma}_{ij}) = +\infty \]

End For

// Creation of the initial events
For each vehicle $V_l \in V$ Do

- If the first task $^{(a)}\tilde{\sigma}_{ij}$ of $V_l$ is an available one Then
  - Compute $T(\tilde{\sigma}_{ij})$ and create event $[V_l, T(\tilde{\sigma}_{ij})]$
End If

End For

// Main loop
While it exists some events $[V_l, T(\tilde{\sigma}_{ij})]$ with $T(\tilde{\sigma}_{ij}) < +\infty$ Do

- Choose the next event $[V_l, T(\tilde{\sigma}_{ij})]$ and compute $T(\sigma_{ij})$
- If the next task $\tilde{\sigma}_{ij'}$ of $V_l$ exists and is available Then $^{(b)}$
  - Compute $T(\tilde{\sigma}_{ij'})$ and create event $[V_l, T(\tilde{\sigma}_{ij'})]$
End If

- If the task $\tilde{\sigma}_{ij+1}$ exists and is the next task of a vehicle $V_l \neq V_l$ Then $^{(c)}$
  - Compute $T(\tilde{\sigma}_{ij+1})$ and create event $[V_l, T(\tilde{\sigma}_{ij+1})]$
End If

End While

Completion times must satisfy the following constraints:

1. $T(\tilde{\sigma}_{i1}) \geq t(M_0, \mu_{i1}), \ \forall J_i \in J$
2. $T(\tilde{\sigma}_{ij}) \geq T(\sigma_{ij-1}) + t(\mu_{ij-1}, \mu_{ij}), \ \forall \tilde{\sigma}_{ij} \in \tilde{O}, \ j > 1$
(3) \[ T(\tilde{\sigma}_{i,j}) \geq T(\tilde{\sigma}_{i,j}) + t'(\mu_{i,j}, \mu_{i,j-1}) + t(\mu_{i,j-1}, \mu_{i,j}), \quad \forall \tilde{\sigma}_{i,j}, \tilde{\sigma}_{i,j} \in \tilde{O}, \]

\[ v(\tilde{\sigma}_{i,j}) = v(\tilde{\sigma}_{i,j}), \lambda(\tilde{\sigma}_{i,j}) = \lambda(\tilde{\sigma}_{i,j}) + 1 \quad \text{(with the convention } \mu_0 = M_0) \]

(4) \[ T(\sigma_{i,j}) \geq T(\sigma_{i,j}) + p_{i,j}, \quad \forall \sigma_{i,j}, \sigma_{i,j} \in O, \mu_{i,j} = \mu_{i,j}, \ T(\tilde{\sigma}_{i,j}) > T(\tilde{\sigma}_{i,j}) \]

(5) \[ T(\sigma_{ij}) \geq T(\tilde{\sigma}_{ij}) + p_{ij}, \quad \forall \sigma_{ij} \in O \]

Constraint 1 indicates that each job has to be transported from the loading station to its first machine. Constraint 2 indicates that a task cannot be completed before its previous corresponding operation has been completed plus the loaded travel time between the two machines. Constraint 3 gives the minimal time between two consecutive transportation tasks (on the same vehicle). The vehicle makes the empty travel from its last delivery destination to its next pick-up destination, plus the loaded travel from here to its next delivery destination. Constraint 4 indicates that an operation can only begin after all previous operations on the same machine have been completed. Constraint 5 indicates that an operation cannot begin before the job arrives on the machine.

Also, the constraint 4 means that the order of operations on a machine is given by the order of arriving corresponding tasks on this machine. In other words, the machine’s input buffer follows the FIFO rule. FIFO is a good performing rule (although nothing indicates that it exists as an optimal solution satisfying this rule) and has some practical applications. This additional assumption allows us to simplify the computation of the makespan. Only the completion time of the previous operation has to be considered.

Moreover, we assume that the task schedule follows a no-delay strategy (an event occurs as soon as possible). So, the completion times are equal to the minimal value satisfying constraints (1–5). The makespan value is equal to the latest completion time.

Some schedules can lead to situations of deadlock. Such situations are presented on figures 1(a) (deadlock with one vehicle) and (b) (deadlock with several vehicles) for jobset 1.

| Vehicles | Task schedule |
|----------|---------------|
| AGV1     | \( \tilde{o}_{11} \) | \( \tilde{o}_{33} \) | \( \tilde{o}_{32} \) | \( \tilde{o}_{14} \) | \( \tilde{o}_{23} \) | \( \tilde{o}_{52} \) |
| AGV2     | \( \tilde{o}_{31} \) | \( \tilde{o}_{42} \) | \( \tilde{o}_{13} \) | \( \tilde{o}_{41} \) | \( \tilde{o}_{22} \) |
| AGV3     | \( \tilde{o}_{21} \) | \( \tilde{o}_{12} \) | \( \tilde{o}_{51} \) | \( \tilde{o}_{43} \) | \( \tilde{o}_{24} \) | \( \tilde{o}_{34} \) | \( \tilde{o}_{53} \) |

(a)

| Vehicles | Task schedule |
|----------|---------------|
| AGV1     | \( \tilde{o}_{51} \) | \( \tilde{o}_{32} \) | \( \tilde{o}_{11} \) | \( \tilde{o}_{14} \) | \( \tilde{o}_{23} \) | \( \tilde{o}_{52} \) |
| AGV2     | \( \tilde{o}_{21} \) | \( \tilde{o}_{42} \) | \( \tilde{o}_{31} \) | \( \tilde{o}_{13} \) | \( \tilde{o}_{33} \) |
| AGV3     | \( \tilde{o}_{22} \) | \( \tilde{o}_{12} \) | \( \tilde{o}_{41} \) | \( \tilde{o}_{43} \) | \( \tilde{o}_{24} \) | \( \tilde{o}_{34} \) | \( \tilde{o}_{53} \) |

(b)

Figure 1. (a) Unfeasible solution: deadlock with one vehicle; (b) Unfeasible solution: deadlock with several vehicles.
Figure 1(a) shows a deadlock with one vehicle. The task \( o_{33} \) cannot be accomplished before the task \( o_{32} \). Figure 1(b) shows a deadlock with three vehicles. Each vehicle waits for accomplishing their second task (respectively \( o_{32}, o_{42} \) and \( o_{12} \)), that another vehicle complete a task scheduled in third position (respectively \( o_{31}, o_{41} \) and \( o_{11} \)).

Deadlock situations are easily detected by our makespan evaluation. Since some tasks cannot be completed, their completion time remains infinite, and so the makespan too.

The presence of unfeasible solutions doesn’t facilitate the good exploration of the search space. However it is not the case here, because it was proved that every feasible solution remains reachable from every other feasible solution into the restricted subset of feasible solutions (Deroussi 2002).

4. Construction of an efficient neighbouring system

A neighbouring system is an essential component of metaheuristics. The efficiency of a neighbouring system can be defined as its faculty to find near-optimal solutions in the shortest possible time. In this section, we describe first a neighbouring system composed of several basic moves. It is recognized that combining several moves permits us to improve the quality of the obtained local minima (Mladenovic and Hansen 1997, Hansen and Mladenovic 2003). Then, a technique is proposed to shrink the neighbourhood size, and so to reduce its exploration at the most promising neighbours. To conclude, summarized results are reported showing the efficiency of the proposed neighbouring system.

4.1 Description of the basic moves

Tian et al. (1999) present some usual basic moves for scheduling problems. Amongst them, we choose exchange moves and insertion moves, because their effectiveness for shop problems has been proved.

4.1.1 Exchange move. This move consists in permuting two beforehand selected tasks. The exchange moves thus defined can be intra-vehicles (exchanged tasks are on the same vehicle) or inter-vehicles (exchanged tasks are on different vehicles). Figure 2 shows an inter-vehicles exchange move.

The exchange move preserves the number of tasks assigned to each vehicle. It results from the fact that local search methods based on this move are strongly dependent on the structure of the initial solution.

This move allows to generate \( o(o - 1)/2 \) neighbours.

4.1.2 Insertion move. The insertion move consists in removing a randomly chosen task from the schedule, and then inserting it at a randomly chosen location (figure 3). Like the exchange moves, the insertion moves can be intra-vehicles or inter-vehicles. However, in the case of inter-vehicles moves, the number of tasks assigned to the vehicles may vary.
The number of solutions that may be generated starting from the same schedule is given by $\frac{o}{C_0^2 + k}$.

4.2 Neighbourhood reduction

Neighbourhood reduction consists in selecting, among the set of neighbours of a given solution and according to a given criterion, a subset of neighbours containing those that are considered as the *a priori* most promising ones. Consequently, it is expected that reduction allows acceleration of the neighbourhood exploration.

We propose to determine the most promising neighbours by considering only (exchange or insertion) moves in which the chosen tasks are completed in a close interval of time. The implemented neighbourhood reduction is described in figure 4.

Once the first operation $\sigma_{i,j_1}$ is chosen, the only moves considered are insertions or exchanges, for which the second operation $\sigma_{i,j_2}$ will satisfy $T(\sigma_{i,j_2}) \leq T(\sigma_{i,j_1})$ with the following convention:

- $T(\sigma_{i,0}) = 0$ if $\sigma_{i,j_1}$ is the first operation in the sequence of $J_i$,
- $T(\sigma_{i,j_1}) = +\infty$ if $\sigma_{i,j_1}$ is the last operation in the sequence of $J_i$.

| Vehicles | Task schedule |
|----------|---------------|
| AGV1     | $\tilde{o}_{11}$ | $\tilde{o}_{31}$ | $\tilde{o}_{22}$ | $\tilde{o}_{14}$ | $\tilde{o}_{23}$ | $\tilde{o}_{52}$ |
| AGV2     | $\tilde{o}_{21}$ | $\tilde{o}_{42}$ | $\tilde{o}_{13}$ | $\tilde{o}_{24}$ | $\tilde{o}_{33}$ |
| AGV3     | $\tilde{o}_{41}$ | $\tilde{o}_{12}$ | $\tilde{o}_{51}$ | $\tilde{o}_{43}$ | $\tilde{o}_{24}$ | $\tilde{o}_{34}$ | $\tilde{o}_{53}$ |

Figure 2. Illustration of the exchange move; tasks $\tilde{o}_{22}$ and $\tilde{o}_{24}$ are exchanged.

| Vehicles | Task schedule |
|----------|---------------|
| AGV1     | $\tilde{o}_{11}$ | $\tilde{o}_{31}$ | $\tilde{o}_{24}$ | $\tilde{o}_{14}$ | $\tilde{o}_{23}$ | $\tilde{o}_{52}$ |
| AGV2     | $\tilde{o}_{21}$ | $\tilde{o}_{42}$ | $\tilde{o}_{13}$ | $\tilde{o}_{22}$ | $\tilde{o}_{33}$ |
| AGV3     | $\tilde{o}_{41}$ | $\tilde{o}_{12}$ | $\tilde{o}_{51}$ | $\tilde{o}_{43}$ | $\tilde{o}_{24}$ | $\tilde{o}_{34}$ | $\tilde{o}_{53}$ |

Figure 3. Illustration of the insertion move; task $\tilde{o}_{24}$ is inserted at third position on AGV1.

The number of solutions that may be generated starting from the same schedule is given by $o(o - 2 + k)$.  

| Vehicles | Task schedule |
|----------|---------------|
| AGV1     | $\tilde{o}_{11}$ | $\tilde{o}_{31}$ | $\tilde{o}_{24}$ | $\tilde{o}_{22}$ | $\tilde{o}_{14}$ | $\tilde{o}_{23}$ | $\tilde{o}_{52}$ |
| AGV2     | $\tilde{o}_{21}$ | $\tilde{o}_{42}$ | $\tilde{o}_{13}$ | $\tilde{o}_{24}$ | $\tilde{o}_{33}$ |
| AGV3     | $\tilde{o}_{41}$ | $\tilde{o}_{12}$ | $\tilde{o}_{51}$ | $\tilde{o}_{43}$ | $\tilde{o}_{24}$ | $\tilde{o}_{34}$ | $\tilde{o}_{53}$ |
4.3 Summary of the results

We build a neighbouring system by considering two basic moves with a reduction of the neighbourhood size. In this section, we verify the efficiency of this neighbouring system. For this, we compute the average value of 1000 executions of each of the 40 instances that compose our benchmark test. One execution consists in a local search method, starting from a randomly generated feasible solution (using the stochastic alternative of the construction heuristic described in section 5.1.). The duration of an execution is evaluated by counting the calls of the makespan evaluation. The results obtained are summarized in table 2. The column ‘Average cost’ indicates the average cost of the obtained local minima. The column ‘Number of evaluations’ gives the average number of makespan computations necessary for obtaining a local minimum.

First, we can see that the neighbourhood reduction significantly improves the behaviour of local search methods. Thus, for the exchange move, a local minimum is obtained three times more quickly with the same quality of local minima. For the insertion move, time is divided by a factor of 5, while a weak loss of quality (1.2%) is noted.

Concerning the combination of reduced basic moves, the quality of the local minima is improved by 7.5% compared to the exchange move, and this in a similar duration. The improvement of quality compared to the insertion move is 4% with a time divided by 2. So, the combination of reduced moves allows us to improve the quality of local minima. Our objective is now to include this neighbouring system into several metaheuristics, and to evaluate the efficiency of their behaviour.

5. Description of the resolution methods

We focus in this section on the resolution methods used. We present first the construction heuristic we have developed, and then the metaheuristics according to the Blum and Roli (2003) formulation.
5.1 Construction heuristic

This greedy heuristic uses the available tasks earlier defined. The basic principle is to successively schedule each available task by assigning it to a vehicle. When a task $\tilde{\sigma}_{ij}$ is completed, the set of the available tasks is updated, by replacing $\tilde{\sigma}_{ij}$ by its successor $\tilde{\sigma}_{ij+1}$, if it exists. The algorithm 2 describes the construction heuristic.

**Algorithm 2: Construction heuristic**

Let $D = \{\tilde{\sigma}_{i1}, i = 1, \ldots, n\}$ be the initial set of available tasks.

While $D \neq \emptyset$ Do

Choose a task $\tilde{\sigma}_{ij} \in D$,

Assign $\tilde{\sigma}_{ij}$ to a vehicle $k \in K$,

Let $D := D/(\tilde{\sigma}_{ij})$,

If it exist let $D := D \cup \{\tilde{\sigma}_{ij+1}\}$

End While

Two alternatives are given for the choice of the task $\tilde{\sigma}_{ij}$ and the vehicle $k$. The first alternative (the stochastic one) builds a random feasible solution, while the second alternative (the deterministic one) tries to build a good feasible solution.

- The task is randomly chosen in $D$ and is accomplished by a randomly chosen vehicle;
- the selected task is the one corresponding to the job having the largest remaining processing time (LPT rule) and the chosen vehicle is the one that will accomplish this task in the shortest time.

5.2 Iterated local search methods

Iterated Local Search methods (ILS) were the subject of interesting development (Lourenço et al. 2003). The principle algorithm of ILS is described in algorithm 3.

The method that we have implemented is one of the simplest forms of ILS. The initial solution (GenerateInitialSolution) is given by the construction heuristic (deterministic alternative) described in the previous section. The local search (LocalSearch) is a variable neighbourhood descent (VND) composed of the reduced exchange and insertion moves. To perturb a solution (perturbation), three successive exchange moves are applied, while making sure that each of them preserves the feasibility of the current solution. The acceptance criterion (ApplyAcceptanceCriterion) consists in accepting the new local minimum $s^*$ if and only if it is better or equal than $s^*$ (Best Walk Criterion). The stopping criterion is a maximum number of iterations arbitrarily fixed at 1000.

**Algorithm 3: Iterated local search methods (ILS)**

$s_0 \leftarrow \text{GenerateInitialSolution}()$

$s^* \leftarrow \text{LocalSearch}(s_0)$

While stopping criterion is not met Do

$s' \leftarrow \text{Perturbation}(s^*, \text{history})$

$s^* \leftarrow \text{LocalSearch}(s')$

$s^* \leftarrow \text{ApplyAcceptanceCriterion}(s^*, s', \text{history})$

End While
5.3 Simulated annealing

Simulated annealing (SA) is recognized to be one of the oldest metaheuristics (Kirkpatrick et al. 1983). Its principle algorithm is given in algorithm 4.

Algorithm 4: Simulated annealing (SA)

\[
\begin{align*}
& s \leftarrow \text{GenerateInitialSolution()} \\
& T \leftarrow T_0 \\
& \text{While stopping criterion is not met Do} \\
& \quad \text{Randomly choose } s' \text{ in the neighbourhood of } s \\
& \quad \text{If } s' \text{ is better than } s \text{ Then} \\
& \quad \quad s \leftarrow s' \\
& \quad \text{Else} \\
& \quad \quad s \leftarrow s' \text{ according to a probability } p(T, s, s') \\
& \text{End If} \\
& \quad \text{Update}(T) \\
& \text{End While}
\end{align*}
\]

We recall that our main goal is to verify the efficiency of the proposed neighbouring system. As a consequence, no study of the parameters space has been investigated, and we choose an empirical tuning of parameters. This choice is based on a basic cooling schedule; the temperature decreases at each iteration (as suggested first by Lundy and Mees 1986) following a geometric law. This allows us to determine the other parameters by considering some simple rules (Van Laarhoven and Aarts 1987) as indicated just below.

An initial solution is randomly generated by the construction heuristic (stochastic alternative). In order for the final solution to be independent from the starting point, the initial temperature must be hot enough. So, the parameter \( T \) that simulates the temperature is initialized with \( T_0 = 5 \) (we have then a probability of 0.5 for accepting a deterioration of the current solution of \( T_0 \times \ln(2) \) at the beginning of the cooling schedule). The stopping temperature is arbitrarily fixed at \( T_a = 10^{-3} \) (cold enough to consider that the probability for accepting a worse transition is null). As mentioned above, the temperature is a geometric sequence of common ratio \( T_f \). \( T_f \) is calculated so as to make a given number of iterations, which is calculated so that the same time has passed for ILS and SA methods. The probability of accepting a worse transition is given by the Boltzmann factor \( \exp(-\Delta E/T) \).

Thus tuned, experimental behaviour of simulated annealing is proven to be quite good.

5.4 Hybrid method simulated annealing/local search

This hybrid method consists of applying the simulated annealing, but on the (very restricted) subspace of local minima rather than to the whole space of solutions. This technique was initially proposed by Martin and Otto (1996) and Desai and Patil (1996). By analogy with Genetic Local Search (GLS) methods, we will use the term ‘Simulated Annealing Local Search’ (SALS) to indicate this method. The principle algorithm of SALS is contained in algorithm 3, in which the acceptance criterion is defined by the SA rules.
The adjustments are the same ones as for simulated annealing, except for the choice of the common ratio $T_f$, which is calculated so as to carry out 1000 iterations (the same ones as for the ILS method).

6. Experiments and computational results

After the description of the benchmark test, we present the results obtained by metaheuristics. A comparison with the literature results is provided in the third part of this section.

6.1 Description of the benchmark test

We use the benchmark test suggested by Bilge and Ulusoy (1995). It is composed of 40 instances. All of them are made up of one LU station, four machines and two vehicles. The loading/unloading times are neglected compared to the transportation times. Empty moves and transportation moves are supposed to spend the same amount of time.

These instances are generated according to ten jobsets, combined with four different topologies for the workshops (see appendices A and B). Topologies are assumed to be representative of existing systems, and the jobsets include between 5 and 8 jobs, and between 13 and 21 operations to be scheduled. The name given to the instances is composed by the prefix ‘Ex’, followed by two numbers representing, respectively, the jobset and the topology. Thus, Ex53 indicates the instance generated by jobset 5 and topology 3.

Moreover, we will consider two criteria for the objective function:

- C1: to minimize the exit time of the last job of the system;
- C2: to minimize the completion time of the last job (makespan).

The C2 criterion is used in the literature and we consider it for comparison purpose. The C1 criterion takes into account, in the optimization process, the return of the vehicles to the unloading station at the end of the schedule. Let us notice that there is a close link between these two criteria. Indeed, they both seek to minimize the completion time of the last task carried out. C1 criterion consider all the tasks $[\bar{\sigma}_{ij} \in \bar{O}, i = 1, \ldots, n, j = 1, \ldots, n_i + 1]$, while C2 consider only the tasks corresponding to non-fictitious operations $[\bar{\sigma}_{ij} \in \bar{O}, i = 1, \ldots, n, j = 1, \ldots, n_i]$. It will be expected that C1 would be more difficult, since it considers more tasks, and it is thus of higher combinatorial.

Although these instances are of relatively low size (13–21 tasks to be scheduled for C2 and 18–27 tasks for C1), they remain very difficult to solve. To date, few instances were solved in an exact way, and the best published lower bounds are of poor quality (Ulusoy et al. 1997).

The best known solutions are summarised in table 3. They will be useful mainly as comparative data for the results that we present. Programs are written in language C, and tested on a Pentium 4, 3.4 GHz. CPU times are between one and four seconds per algorithm execution.
6.2 Comparison of the studied methods

Table 4 contains the results obtained by the deterministic construction heuristic for each instance and each criterion. Column ‘res’ indicates the makespan obtained, while the column ‘%’ gives the relative gap compared to the best known upper bound (table 3).

The construction heuristic provides a mean deviation from the value of the best known solution of 36% for the C1 criterion and of 28% for the C2 criterion. These results clearly show the relative weakness of the heuristic, and the necessity of improving the obtained solutions with metaheuristics.

The metaheuristics results are presented in tables 5(a) and (b) (columns ‘ILS’, ‘SA’ and ‘SALS’ gives the results of the corresponding method, while the column ‘Time’ indicates the CPU time in seconds for one algorithm execution). Table 5(a) contains results for C1 criterion and table 5(b) for C2 criterion. Values in bold type indicate that the corresponding method has found the best known upper bound. Furthermore, because SA and SALS are stochastic algorithms, 10 replications are carried out for each instance so as to limit the random influence on the results. Tables 5 give the results obtained by the best replication.

In a general way, the results obtained are homogeneous. As an example, the three methods give the same results for 34 of the 40 instances tested for the C1 criterion. In addition, ILS, SA and SALS find the best known upper bound, respectively, for 35, 37 and all instances. More scattered results are obtained for C2 criterion.
In order to have more information about the robustness of SA and SALS, we give in table 6 the best, the worst and the average of all the replications for the two criteria. We can see that most of the execution of SALS leads to very good solutions (on average, a replication is at 0.34% of the best obtained replication for the C2 criterion, and 0.96% for the C1 criterion), while the results of SA are more scattered (we obtain, respectively, 1.47% for C2 and 2.67% for C1). If the best replications of SALS and SA are close (respectively 108.48 against 108.63 for C2 criterion, 124.00 against 124.38 for C1), there is a significant difference about their average replication (108.85 against 110.23 for C2 criterion, 125.19 against 127.70 for C1).

So, we can conclude that SALS is an effective method for the studied problem. In the next section, we compare it with methods proposed in the literature.

### 6.3 Comparison with the literature

Table 7 summarizes the best results obtained for C2 criterion (the only one studied in the literature):

- (i) By a synthesis of three papers (Ülusoy and Bilge 1993, Bilge and Ülusoy 1995 and Ülusoy et al. 1997) (B&U);
- (ii) by the genetic algorithm of Abdelmaguid et al. (2004) (GAA);
- (iii) by our hybrid metaheuristic SALS.

---

### Table 4. Results obtained with the construction heuristic.

| Instances | C1   |   | C2   |   | Instances | C1   |   | C2   |   |
|-----------|------|---|------|---|-----------|------|---|------|---|
|           | res  | % | res  | % |           | res  | % | res  | % |
| Ex11      | 152  | 33.33 | 114  | 18.75 | Ex61      | 175  | 35.66 | 138  | 16.95 |
| Ex12      | 106  | 17.78 | 90   | 9.76  | Ex62      | 136  | 33.33 | 122  | 24.49 |
| Ex13      | 110  | 12.24 | 88   | 4.76  | Ex63      | 147  | 40.00 | 124  | 20.39 |
| Ex14      | 174  | 24.29 | 124  | 20.39 | Ex64      | 213  | 41.06 | 143  | 19.17 |
| Ex21      | 154  | 32.76 | 136  | 36.00 | Ex71      | 200  | 50.38 | 140  | 26.13 |
| Ex22      | 122  | 48.78 | 118  | 55.26 | Ex72      | 136  | 58.14 | 110  | 39.24 |
| Ex23      | 134  | 50.56 | 128  | 48.84 | Ex73      | 149  | 60.22 | 111  | 33.73 |
| Ex24      | 202  | 50.75 | 160  | 48.15 | Ex74      | 258  | 60.25 | 160  | 26.98 |
| Ex31      | 158  | 30.58 | 120  | 21.21 | Ex81      | 203  | 21.56 | 197  | 22.36 |
| Ex32      | 108  | 21.35 | 94   | 10.59 | Ex82      | 181  | 16.77 | 177  | 17.22 |
| Ex33      | 116  | 20.83 | 96   | 11.63 | Ex83      | 185  | 19.35 | 183  | 19.61 |
| Ex34      | 190  | 28.38 | 134  | 20.72 | Ex84      | 252  | 41.57 | 219  | 34.36 |
| Ex41      | 194  | 40.58 | 156  | 39.29 | Ex91      | 170  | 33.86 | 141  | 21.55 |
| Ex42      | 150  | 50.00 | 127  | 45.98 | Ex92      | 139  | 31.13 | 125  | 22.55 |
| Ex43      | 161  | 57.84 | 131  | 47.19 | Ex93      | 144  | 34.58 | 127  | 20.95 |
| Ex44      | 227  | 39.26 | 175  | 44.63 | Ex94      | 211  | 41.61 | 154  | 28.33 |
| Ex51      | 146  | 32.73 | 112  | 28.74 | Ex101     | 208  | 35.95 | 184  | 25.17 |
| Ex52      | 103  | 27.16 | 92   | 33.33 | Ex102     | 184  | 32.37 | 180  | 33.33 |
| Ex53      | 117  | 31.46 | 100  | 35.14 | Ex103     | 195  | 39.29 | 182  | 31.88 |
| Ex54      | 172  | 28.36 | 134  | 39.58 | Ex104     | 249  | 36.07 | 199  | 25.16 |

(the best known upper bound is reached, respectively, 22, 28 and 36 times by ILS, SA and SALS). It seems that the additional tasks make effectively the instances harder to solve.
### Table 5(a). Comparison of the three studied metaheuristics (C1 criterion).

| Instances | ILS  | SA   | SALS  | Time (s) | Instances | ILS  | SA   | SALS  | Time (s) |
|-----------|------|------|-------|----------|-----------|------|------|-------|----------|
| Ex11      | 114  | 114  | 114   | 1.3      | Ex61      | 130  | 129  | 129   | 2.7      |
| Ex12      | 90   | 90   | 90    | 1.4      | Ex62      | 102  | 102  | 102   | 2.5      |
| Ex13      | 98   | 98   | 98    | 1.4      | Ex63      | 106  | 105  | 105   | 2.6      |
| Ex14      | 140  | 140  | 140   | 1.3      | Ex64      | 151  | 151  | 153   | 2.6      |
| Ex21      | 118  | 118  | 116   | 2.3      | Ex71      | 134  | 137  | 135   | 3.9      |
| Ex22      | 82   | 82   | 82    | 2.3      | Ex72      | 90   | 86   | 86    | 4.0      |
| Ex23      | 90   | 90   | 89    | 2.2      | Ex73      | 98   | 93   | 93    | 3.7      |
| Ex24      | 138  | 136  | 134   | 2.1      | Ex74      | 167  | 163  | 161   | 3.8      |
| Ex31      | 121  | 121  | 121   | 2.2      | Ex81      | 167  | 167  | 167   | 3.0      |
| Ex32      | 89   | 89   | 89    | 2.2      | Ex82      | 155  | 155  | 155   | 2.4      |
| Ex33      | 96   | 96   | 96    | 2.1      | Ex83      | 155  | 155  | 155   | 2.4      |
| Ex34      | 149  | 148  | 148   | 2.1      | Ex84      | 178  | 178  | 178   | 3.3      |
| Ex41      | 138  | 138  | 138   | 2.1      | Ex91      | 129  | 129  | 129   | 1.9      |
| Ex42      | 102  | 100  | 100   | 2.2      | Ex92      | 106  | 106  | 106   | 1.9      |
| Ex43      | 106  | 104  | 102   | 2.1      | Ex93      | 107  | 107  | 107   | 1.9      |
| Ex44      | 164  | 163  | 163   | 2.3      | Ex94      | 149  | 150  | 149   | 1.9      |
| Ex51      | 110  | 110  | 110   | 1.3      | Ex101     | 156  | 153  | 153   | 3.3      |
| Ex52      | 81   | 83   | 81    | 1.4      | Ex102     | 140  | 139  | 139   | 3.1      |
| Ex53      | 89   | 89   | 89    | 1.4      | Ex103     | 141  | 141  | 141   | 3.0      |
| Ex54      | 134  | 136  | 134   | 1.3      | Ex104     | 187  | 184  | 183   | 3.3      |

The * symbol indicates new upper bounds.

### Table 5(b). Comparison of the three studied metaheuristics (C2 criterion).

| Instances | ILS  | SA   | SALS  | Time (s) | Instances | ILS  | SA   | SALS  | Time (s) |
|-----------|------|------|-------|----------|-----------|------|------|-------|----------|
| Ex11      | 96   | 96   | 96    | 1.3      | Ex61      | 118  | 118  | 118   | 2.5      |
| Ex12      | 82   | 82   | 82    | 1.2      | Ex62      | 98   | 98   | 98    | 2.3      |
| Ex13      | 84   | 84   | 84    | 1.3      | Ex63      | 103  | 103  | 103*  | 2.4      |
| Ex14      | 103  | 103  | 103   | 1.3      | Ex64      | 120  | 120  | 120   | 2.5      |
| Ex21      | 100* | 100* | 100*  | 2.1      | Ex71      | 111* | 111* | 111*  | 3.9      |
| Ex22      | 76   | 76   | 76    | 2.0      | Ex72      | 79   | 79   | 79    | 3.8      |
| Ex23      | 86   | 86   | 86    | 2.0      | Ex73      | 84*  | 83*  | 83*   | 3.6      |
| Ex24      | 108  | 108  | 108   | 2.1      | Ex74      | 126* | 126* | 126*  | 3.9      |
| Ex31      | 99   | 99   | 99    | 2.0      | Ex81      | 161  | 161  | 161   | 2.6      |
| Ex32      | 85   | 85   | 85    | 1.9      | Ex82      | 151  | 151  | 151   | 2.3      |
| Ex33      | 86   | 86   | 86    | 1.9      | Ex83      | 153  | 153  | 153   | 2.3      |
| Ex34      | 111  | 111  | 111   | 2.1      | Ex84      | 163  | 163  | 163   | 2.9      |
| Ex41      | 112  | 112  | 112   | 2.1      | Ex91      | 116* | 116* | 116*  | 1.8      |
| Ex42      | 88   | 88   | 87*   | 2.1      | Ex92      | 102  | 102  | 102   | 1.8      |
| Ex43      | 89   | 89   | 89    | 2.1      | Ex93      | 105  | 105  | 105   | 1.8      |
| Ex44      | 121* | 125  | 121*  | 2.0      | Ex94      | 120* | 120* | 120*  | 1.8      |
| Ex51      | 87   | 87   | 87    | 1.3      | Ex101     | 148  | 147  | 147   | 3.0      |
| Ex52      | 69   | 69   | 69    | 1.3      | Ex102     | 135* | 135* | 135*  | 2.9      |
| Ex53      | 74   | 74   | 74    | 1.3      | Ex103     | 139  | 139  | 138*  | 2.9      |
| Ex54      | 96   | 96   | 96    | 1.3      | Ex104     | 160  | 159  | 159   | 3.1      |

The * symbol indicates new upper bounds.
A performance criterion indicates, for each method, the relative gap compared to the best published upper bound (indicated in bold type). The new upper bounds we have obtained are marked with the * symbol.

These results clearly show the superiority of SALS. Our method always finds a result at least as good as those published so far. Moreover, new upper bounds are obtained for all instances.
found for 11 of the 40 instances. SALS finds better solutions than B&U for 23 instances, and than GAA for 13 instances. The improvements are sometimes significant; a solution of cost 111 is obtained against 115 for Ex71 instance, that is to say a relative gain of 3.6% (figure 5 shows, using a Gantt chart representation, the solutions obtained for Ex71 with each of the two criteria).

Still more than quality of the solutions obtained, it is the frequency of obtaining the best results which is most impressive, and this in extremely reasonable computing times for an optimisation problem (about a few seconds). The behaviour of the method suggests that the proposed solutions must be for many of the optimal ones. This remark still increases the interest of the C1 criterion, which makes the instances more difficult to solve. Even if there are only few additional transports to schedule, we notice that SALS loses in robustness for these 'new' instances (see table 6), and does not systematically find the best solutions (for at least four instances). These results might constitute a new reference for future studies, more able to measure the quality of the methods.

7. Conclusion and further works

In this paper, we propose a new effective representation solution for the job-shop problem with transportation. A powerful neighbouring system is described and implemented with three different metaheuristics. The results obtained clearly show the relevance of this approach. Whatever the method used, the presented results are always better or at least equal to those of the literature. One of these methods, SALS, combines robustness with quality, and is a particularly efficient method for this problem.
Figures 5(a) and 5(b). Gantt charts of the new best known solutions for Ex71 instance (C1 and C2 criteria). The two Gantt charts show the machining of the eight jobs (numbered from A to H) on the machines (numbered from M1 to M4). On figure 5(a), the unloading station LU points out, for each job, its exit date of the system. The schedule of the vehicles (AGV1 and AGV2) gives the origin machine, the destination machine, and, if not empty, the transported job.
Also, we propose a new criterion for the solution evaluation, which allows us to extend the literature benchmark test with more challenging instances.

Many perspectives can be drawn from this work.

Concerning the studied problem, the benchmark test proposed by Bilge and Ülsoy (1995) seems out-of-date, and it would have to be extended by more difficult instances. Concurrently to this extension, and paradoxically, the exact resolution of most of these instances is still today out of reach, and constitutes an always topical orientation research.

The consideration of online dispatching rules leads to define a more difficult makespan evaluation, which can require the use of discrete event simulation. Combining simulation with optimization methods can be a very interesting perspective to our work.

Lastly, the methods used in this paper are trajectory metaheuristics. The fact of carrying out several replications, and of keeping in memory the solution obtained by the best of them, can be seen like the use of a population method, in which the individuals are private of any possibility of communicating. Restoring this possibility (by using a population method, like a genetic algorithm) may allow us to improve the results proposed in this paper.

Appendices

Appendix A: the jobsets

For each job, the sequence of machines to be visited is given. The processing time is indicated in brackets.

**Jobset 1:**

| Job 1: M1(8); M2(16); M4(12) |
|-----------------------------|
| Job 2: M1(20); M3(10); M2(18) |
| Job 3: M3(12); M4(8); M1(15) |
| Job 4: M4(14); M2(18) |
| Job 5: M3(10); M1(15) |

**Jobset 3:**

| Job 1: M1(16); M3(15) |
|-----------------------|
| Job 2: M2(18); M4(15) |
| Job 3: M1(20); M2(10) |
| Job 4: M3(15); M4(10) |
| Job 5: M1(8); M2(10); M3(15); M4(17) |
| Job 6: M2(10); M3(15); M4(8); M1(15) |

**Jobset 2:**

| Job 1: M1(10); M4(18) |
|-----------------------|
| Job 2: M2(10); M4(18) |
| Job 3: M1(10); M3(20) |
| Job 4: M2(10); M3(15); M4(12) |
| Job 5: M1(10); M2(15); M4(12) |
| Job 6: M1(10); M2(15); M3(12) |

**Jobset 5:**

| Job 1: M1(6); M2(12); M4(9) |
|-----------------------------|
| Job 2: M1(18); M3(6); M2(15) |
| Job 3: M3(9); M4(3); M1(12) |
| Job 4: M4(6); M2(15) |
| Job 5: M3(3); M1(9); |

**Jobset 4:**

| Job 1: M4(11); M1(10); M2(7) |
|-----------------------------|
| Job 2: M3(12); M2(10); M4(8) |
| Job 3: M2(7); M3(10); M1(9); M3(8) |
| Job 4: M2(7); M4(8); M1(12); M2(6) |
| Job 5: M1(9); M2(7); M4(8); |
| M2(10); M3(8) |

**Jobset 6:**

| Job 1: M1(9); M2(11); M4(7) |
|-----------------------------|
| Job 2: M1(19); M2(20); M4(13) |
| Job 3: M2(14); M3(20); M4(9) |
| Job 4: M2(14); M3(20); M4(9) |
| Job 5: M1(11); M3(16); M4(8) |
| Job 6: M1(10); M3(12); M4(10) |
Jobset 7:
Job 1: M1(6); M4(6)
Job 2: M2(11); M4(9)
Job 3: M2(9); M4(7)
Job 4: M3(16); M4(7)
Job 5: M1(9); M3(18)
Job 6: M2(13); M3(19); M4(6)
Job 7: M1(10); M2(9); M3(13)
Job 8: M1(11); M2(9); M4(8)

Jobset 8:
Job 1: M2(12); M3(21); M4(11)
Job 2: M2(12); M3(21); M4(11)
Job 3: M2(12); M3(21); M4(11)
Job 4: M2(12); M3(21); M4(11)
Job 5: M1(10); M2(14); M3(18); M4(9)
Job 6: M1(10); M2(14); M3(18); M4(9)

Jobset 9:
Job 1: M3(9); M1(12); M2(9); M4(6)
Job 2: M3(16); M2(11); M4(9)
Job 3: M1(21); M2(18); M4(7)
Job 4: M2(20); M3(22); M4(11)
Job 5: M3(14); M1(16); M2(13); M4(9)

Jobset 10:
Job 1: M1(11); M3(19); M2(16); M4(13)
Job 2: M2(21); M3(16); M4(14)
Job 3: M3(8); M2(10); M1(14); M4(9)
Job 4: M2(13); M3(20); M4(10)
Job 5: M1(9); M3(16); M4(18)
Job 6: M2(19); M1(21); M3(11); M4(15)

Appendix B: the topologies
The transport times to go from any machine to another one is given in the following matrix:

\[
\begin{array}{cccccc}
\text{FMS1} & \text{LU} & \text{M1} & \text{M2} & \text{M3} & \text{M4} \\
\text{LU} & 0 & 6 & 8 & 10 & 12 \\
\text{M1} & 12 & 0 & 6 & 8 & 10 \\
\text{M2} & 10 & 6 & 0 & 6 & 8 \\
\text{M3} & 8 & 8 & 6 & 0 & 6 \\
\text{M4} & 6 & 10 & 8 & 6 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{FMS2} & \text{LU} & \text{M1} & \text{M2} & \text{M3} & \text{M4} \\
\text{LU} & 0 & 4 & 6 & 8 & 6 \\
\text{M1} & 6 & 0 & 2 & 4 & 2 \\
\text{M2} & 8 & 12 & 0 & 2 & 4 \\
\text{M3} & 6 & 10 & 12 & 0 & 2 \\
\text{M4} & 4 & 8 & 10 & 12 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{FMS3} & \text{LU} & \text{M1} & \text{M2} & \text{M3} & \text{M4} \\
\text{LU} & 0 & 2 & 4 & 10 & 12 \\
\text{M1} & 12 & 0 & 2 & 8 & 10 \\
\text{M2} & 10 & 12 & 0 & 6 & 8 \\
\text{M3} & 4 & 6 & 8 & 0 & 2 \\
\text{M4} & 2 & 4 & 6 & 12 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{FMS4} & \text{LU} & \text{M1} & \text{M2} & \text{M3} & \text{M4} \\
\text{LU} & 0 & 4 & 8 & 10 & 14 \\
\text{M1} & 18 & 0 & 4 & 6 & 10 \\
\text{M2} & 20 & 14 & 0 & 8 & 6 \\
\text{M3} & 12 & 8 & 6 & 0 & 6 \\
\text{M4} & 14 & 14 & 12 & 6 & 0 \\
\end{array}
\]
References

Abdelmaguid, T.F., Nassef, O.N., Kamal, B.A. and Hassan, M.F., A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *Int. J. Prod. Res.*, 2004, **42**, 267–281.

Bilge, Ü. and Ülusuoy, G., A time window approach to simultaneous scheduling of machines and material handling system in a FMS. *Oper. Res.*, 1995, **43**, 1058–1070.

Blazewicz, J., Eiseit, H.A., Finke, G., Laporte, G. and Weglarz, J., Scheduling tasks and vehicles in a flexible manufacturing system. *Int. J. Flex. Manuf. Sys.*, 1991, **4**, 5–16.

Blum, C. and Roli, A., Metaheuristics in combinatorial optimisation: overview and conceptual comparison. *ACM Comput. Surveys*, 2003, **35**, 268–308.

Brucker, P., *Scheduling Algorithms*, 3rd ed., 2001 (Springer-Verlag: Berlin).

Deroussi, L., *Heuristiques, métahéuristiques et systèmes de voisinage: application à des problèmes théoriques et industriels de type TSP et ordonnancement*. PhD thesis (in French), University of Clermont-Ferrand II, 2002.

Desai, R. and Patil, R., SALO: Combining simulated annealing and local optimisation for efficient global optimisation. Technical Report LA-UR. Available online at: http://citeseer.ist.psu.edu/desai96salo.html (accessed 1996).

Egbelu, P.J. and Tanchoco, J.M.A., Characterisation of automated guided vehicle dispatching rules. *Int. J. Prod. Res.*, 1984, **22**, 359–374.

Ganesharajah, T., Hall, N.G. and Srisandarajah, C., Design and operational issues in AGV-served manufacturing systems. *Ann. Oper. Res.*, 1998, **76**, 109–154.

Gobal, S.L. and Kasilingam, R.G., A simulation model for estimating vehicle requirements in automated guided vehicle systems. *Comput. Indus. Engng.*, 1991, **21**, 623–627.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., Optimisation and approximation in deterministic sequencing and scheduling: a survey. *Ann. Disc. Mathemat.*, 1979, **5**, 236–287.

Hansen, P. and Mladenovic, N., Variable neighborhood search. In *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, pp. 145–184, 2003 (Kluwers Academic Publishers: Norwell, MA).

Hurink, J.L. and Knust, S., Tabu A tabu search algorithm for scheduling a single robot in a job-shop environment. *Discrete Applied Mathematics*, 2002, **119**, 181–203.

Kirkpatrick, S., Velatt, C.D. and Vecchi, M.P., Optimisation by simulated annealing. *Science*, 1983, **222**, 671–680.

Knust, S., Shop-scheduling problems with transportation. PhD thesis, Fachbereich Mathematik/Informatik Universität Osnabrück, 1999.

Le-Anh, T. and De Koster, M.B.M., On-line dispatching rules for vehicle-based internal transport systems. *Int. J. Prod. Res.*, 2005, **43**, 1711–1728.

Le-Anh, T. and De Koster, M.B.M., A review of design and control of automated guided vehicle systems. *Eur. J. Oper. Res.*, 2006, **121**, 1–23.

Lenstra, J.K. and Rinnooy Kan, A.H.G., Computational complexity of discrete optimization problems. *Ann. Disc. Mathem.*, 1979, **4**, 121–140.

Lenstra, J.K. and Rinnooy Kan, A.H.G., Complexity of vehicle routing and scheduling problems. *Networks*, 1981, **11**, 221–227.

Lourenço, H.R., Martin, O.C. and Stützle, T., Iterated local search. In *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, pp. 321–353, 2003 (Kluwers Academic Publishers: Norwell).

Lundy, M. and Mees, A., Convergence of an annealing algorithm. *Mathemat. Program.*, 1986, **34**, 111–124.

Martin, O.C. and Otto, S.W., Combining simulated annealing with local search heuristics. *Ann. Oper. Res.*, 1996, **63**, 57–75.

Mladenovic, N. and Hansen, P., Variable neighborhood search. *Comput. Oper. Res.*, 1997, **24**, 1097–1100.

Nuhut, O., Scheduling of automated guided vehicles. Available online at: http://citeseer.its.psu.edu/nuhut99scheduling.html (accessed 1999).

Qiu, L., Hsu, W., Huang, S.Y. and Wang, H., Scheduling and routing algorithms for AGVs: a survey. *Int. J. Prod. Res.*, 2002, **40**, 745–760.
Raman, N., Talbot, F.B. and Rachamadgu, R.V., Simultaneous scheduling of machines and material handling devices in automated manufacturing, in Proceedings of the 2nd ORSA/TIMS Conference on Flexible Manufacturing Systems, 1986, pp. 455–466.

Revielotis, S.A., Conflict resolution in AGV systems. IIE Trans., 2000, 32, 647–659.

Soyle, M., Özdemirel, N.E. and Kayaligil, S., A self-organising neural network approach for the single AGV routing problem. Eur. J. Oper. Res., 2000, 121, 124–137.

Tian, P., Ma, J. and Zhang, D.M., Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: an investigation of generation mechanism. Eur. J. Oper. Res., 1999, 118, 81–94.

Ülusoy, G. and Bilge, Ü., Simultaneous scheduling of machines and automated guided vehicles. Int. J. Prod. Res., 1993, 31, 2857–2873.

Ülusoy, G., Sivrikaya-Serifoglu, F. and Bilge, Ü., A genetic algorithm approach to the simultaneous scheduling of stations and automated guided vehicles. Comput. Oper. Res., 1997, 24, 335–351.

Van Laarhoven, P.J.M. and Aarts, E.H.L., Simulated Annealing: Theory and Applications, 1987 (Kluwer Academic Publishers: Berlin).