Minimizing and balancing setups in a serial production system

Paolo Detti, Marco Pranzo, Carlo Meloni

Minimizing and balancing setups in a serial production system

Journal:	*International Journal of Production Research*
Manuscript ID:	TPRS-2005-IJPR-0090.R1
Manuscript Type:	Original Manuscript
Date Submitted by the Author:	22-Nov-2005
Complete List of Authors:	Detti, Paolo; Università di Siena, Dipartimento di Ingegneria dell'Informazione Pranzo, Marco; Università di Siena, Dipartimento di Ingengeria dell'Informazione Meloni, Carlo; Politecnico di Bari, Dipartimento di Elettrotecnica ed Elettronica
Keywords:	META-HEURISTICS, SEQUENCING, MANUFACTURING MANAGEMENT, SET-UP REDUCTION, OPTIMIZATION
Keywords (user):	bi-objective optimization

Minimizing and balancing setups in a serial production system

Paolo Detti[1], Carlo Meloni[2], Marco Pranzo[1]

[1] Dipartimento di Ingegneria dell'Informazione - Università di Siena, Italy.

{detti,pranzo}@dii.unisi.it

[2] Dipartimento di Elettrotecnica ed Elettronica - Politecnico di Bari, Italy.

meloni@deemail.poliba.it

Abstract

This paper addresses a problem arising in the coordination between two consecutive manufacturing departments of a production system, in which parts are processed in batches, and each batch is characterized by two distinct attributes. Due to limited interstage buffering between the two stages, the two departments have to follow the same batch sequence. In the first department, a setup occurs every time the first attribute of the new batch is different from the previous one. In the downstream department, there is a setup when the second attribute of the new batch changes. The problem consists in finding a common batch sequence optimizing some global utility index. Here we propose a metaheuristic approach to a bi-criteria version of the problem considering two indices, namely the total number of setups paid by the two departments and the maximum number of setups paid by either department.

Keywords: bi-objective optimization, setup, sequencing, metaheuristic, manufacturing

1 Introduction

This paper addresses a coordination problem between two subsequent departments of a production system, in which parts are processed in batches. Due to limited interstage buffering between the two stages, the departments have to process the batches in the same order. Batches to produce are characterized by two different attributes, say A_1 and A_2. In the first (the second) department, a *setup* occurs when the attribute A_1 (attribute A_2) of the next batch to be processed changes. Therefore, a sequence of batches results in a number of setups to be paid by each department, and the problem is finding a batch sequence minimizing the costs related to *setups* (called also changeovers).

In this paper, the case in which all the setups have the same cost is considered (such cases arise in systems in which the activities involved in a changeover are very similar, regardless of the particular changeover) and, hence, the problem consists in finding a batch sequence which both stages will follow to minimize one or more objective functions depending on the number of setups occurred in the two departments. In particular, two objectives have been considered for the problem. Namely, the minimization of the total number of the setups in

the two departments and the minimization of the maximum number of setups paid by either department. While the first objective corresponds to the maximization of the overall utility, the second captures more realistically the need to balance the changeover costs between the departments. The problem is known to be $\mathcal{NP}$-hard even when only one of the above objectives is considered [2].

Our study refers to the real industrial context addressed in [2], in which a large number of different slabs of wood are cut, painted and assembled to build kitchen furniture. In this system, two consecutive departments are considered and no resequencing is possible between the two stages. The two departments are the cutting and the painting department and batches are characterized by a shape and a color. In the cutting department, a setup occurs when a batch has a different shape from the preceding one (cutting tools and machinery must be reconfigured). Similarly, in the painting station a setup occurs when a new color is used (the equipment and the pallets must be thoroughly cleaned in order to eliminate the residuals of the previous color).

For the bi-objective problem, a metaheuristic algorithm able to find in a reasonable computation time a good approximation of the *Pareto-optimal front* is proposed. In general, a solution of the problem is a set of tradeoff solutions, i.e., non-dominated solutions. The paper is organized as follows. In Section 2, literature results and applications are discussed. In Section 3, a formal description of the problem is given. In Section 4, a metaheuristic algorithm, based on an Iterated Local Search approach, is presented, and in Section 5 a large set of experiments is reported showing the effectiveness of the proposed approach. Besides comparisons with two algorithms from the literature are also performed. Finally, in Section 6 conclusions are drawn.

2 Literature and applications

The complexity of the decision processes in the production management involving changeover problems often falls in the class of multicriteria combinatorial optimization problems [30]. In the last years, the topics on multicriteria planning and scheduling problems have been subject to an increasing interest, see for example [5, 13].

Minimizing the impact of changeovers has been widely described as a main component of modern production management strategies [33]. Pursuing high changeover performances is a way to enable agile and responsive manufacturing processes by improving line productivity and reducing downtime losses [23]. This aspect of the production management involving both organizational and economic aspects has received an increasingly attention also in fields as applied mathematics and operations research.

In particular, over the past few decades, there has been a significant effort associated with reducing the time required to perform setups and developing suitable changeover modeling processes. This process can be quite complicated, but yields important benefits in planning and scheduling a production system [33] improving both its production capacity and its manageability. An important survey on changeover problems is proposed in [6]; a wide discussion on MIP models is given in [34]; while [27] offers a review of heuristics for setup problems in serial systems. The models and algorithms presented in this paper particularly refer to the furniture production case addressed in [2]. Recently, two genetic approaches for the problem addressed in this paper have been independently developed. In [25] a genetic algorithm for some bi-objective variants of the problem of sequencing production batches has been proposed,

and in [24] a multi-objective genetic algorithm (MOGA) for the same problem addressed in this paper is implemented, and a computational study is presented.

The studied problem can be also viewed as a special case of the tool-switch problem in a flexible machine [31, 9] in which the tool magazine can accomodate a limited number of elements of two different classes. Several variants on the problem are presented in the literature [8] and different algorithmic approaches are proposed to tackle them [3, 19].

The literature also presents other different real–world applications of some variants of the problem addressed in this paper. As examples, we cite the optimization of the line scheduling for production of components for catalytic converters considered in [29] where the authors propose an hybrid approach based on the integration between Constraint Logic Programming and Genetic Algorithm solving small size instances. The setup sequencing problems arising in the weaving industry are addressed in [4] and a polynomial time algorithm is proposed under some strong assumptions on the cost structure. In [32] a genetic approach for part loading scheduling in a forging machine is addressed. In [10] a decomposition approach to the batching and sequencing problem is considered, while a particular process selection and sequencing in a multi–agent context is addressed in [1].

3 Problem formulation and notation

The problem we consider in this paper, hereinafter called *bi-criteria Setup Coordination* (*bi-SC*), can be formulated as follows. Let E be a set of batches to be produced. The batches must be processed by two departments, called D_1 and D_2, in the same order. Each batch is characterized by two attributes, say for example shape and color. Let V_1 (V_2) denote the set of all possible shapes (colors). We denote the shapes as s_i, with $i = 1, \ldots, |V_1|$, and the colors as c_j, with $j = 1, \ldots, |V_2|$. Each batch is therefore defined by a pair (s_i,c_j). If batch (s_i,c_j) is processed immediately after batch (s_h,c_k), a setup is paid by department D_1 if $s_h \neq s_i$, and a setup is paid by department D_2 if $c_k \neq c_j$.

We can represent the input of the problem as a bipartite graph, $G = (V_1, V_2, E)$, in which nodes in V_1 correspond to shapes, nodes in V_2 to colors and each edge of E corresponds to a batch to be produced. The problem is to sequence the batches in a profitable way from the viewpoint of the number of setups paid by departments. This means that we must find a particular ordering σ of the edges of G. If two consecutive edges (i, j) and (h, k) in σ have no nodes in common, then both departments have to pay one setup when switching from batch (i, j) to batch (h, k). We refer to this case as a *global changeover*. On the other hand, if $i = h$ (if $j = k$), only department D_1 (department D_2) pays a setup, and this is called *local changeover*. Given a sequence σ, we denote as $N_1(\sigma)$ and $N_2(\sigma)$ respectively the number of setups incurred by department D_1 and D_2, respectively. $N_1(\sigma)$ and $N_2(\sigma)$ can be computed as follows. Let $\delta_{i,h}$ be equal to 1 if $i \neq h$ and 0 otherwise, let $s(\sigma(q))$ denote the shape of the q-th batch in the sequence σ, and let $c(\sigma(q))$ denote its color. Then,

$$N_1(\sigma) = 1 + \sum_{q=1}^{|E|-1} \delta_{s(\sigma(q)),s(\sigma(q+1))} \tag{1}$$

$$N_2(\sigma) = 1 + \sum_{q=1}^{|E|-1} \delta_{c(\sigma(q)),c(\sigma(q+1))} \tag{2}$$

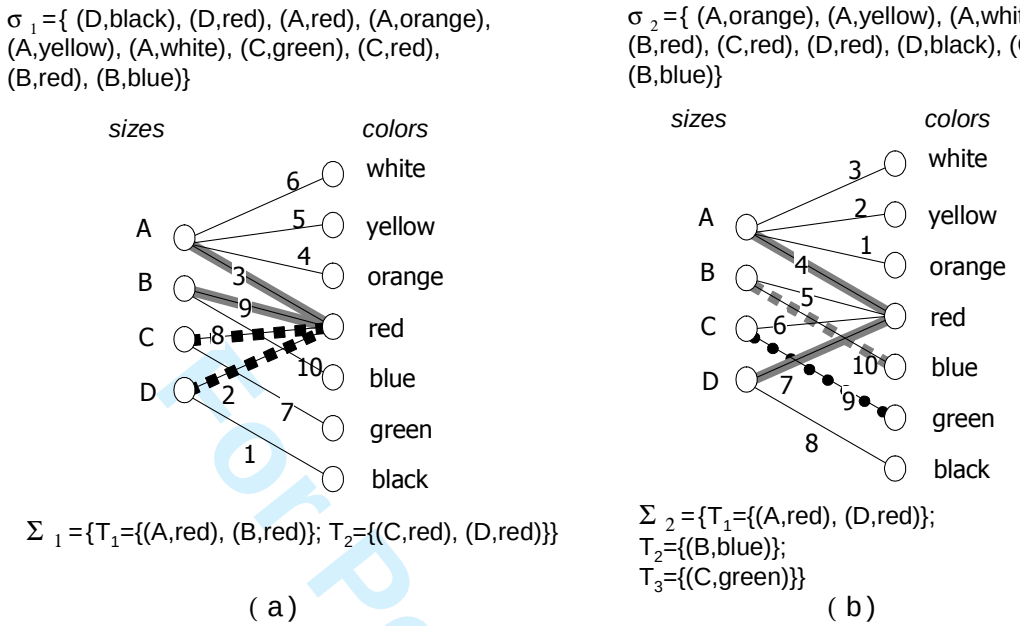

Figure 1: Optimal solutions for a sample instance, with $|V_1| = 4$, $|V_2| = 7$ and $|E| = 10$, of MINSUM (a) and MINMAX (b).

Given a bipartite graph $G = (V_1, V_2, E)$, the two objectives that have been considered can be formally defined as follows:

$$\text{MINSUM} = \min\{N_1(\sigma) + N_2(\sigma)\}.$$

$$\text{MINMAX} = \min\{\max\{N_1(\sigma), N_2(\sigma)\}\}.$$

Hereinafter, where it causes no ambiguities, we refer to MINSUM (MINMAX) to denote both the objective function and the single objective problem of minimizing MINSUM (MINMAX).

It is easy to see that the objectives MINSUM and MINMAX are not equivalent. In fact, consider the example in Figure 1, where 10 batches have to be produced, with 4 shapes and 7 colors. In Figure 1(a), σ_1 is the optimal edge sequence for MINSUM, where $N_1(\sigma)+N_2(\sigma) = 12$ and $\max\{N_1(\sigma), N_2(\sigma)\} = 8$. In Figure 1(b) σ_2 is the optimal edge sequence for MINMAX, where $N_1(\sigma) + N_2(\sigma) = 13$ and $\max\{N_1(\sigma), N_2(\sigma)\} = 7$. In Figures 1(a) and 1(b) the edge of sequences σ_1 and σ_2 are also specified by the numbers associated to the edges. The two optimal solutions have been found by complete enumeration of the solution space.

It is important to observe that, good solutions respect to the two objectives can be achieved, in general, by minimizing the number of setups in the two departments, i.e., $N_1(\sigma)$ and $N_2(\sigma)$. In fact, the two considered criteria are strongly correlated since they both depend on the values $N_1(\sigma)$ and $N_2(\sigma)$. Nevertheless, as shown by the above example there is not a unique function suitable to be used as scalar optimization objective. However, the correlation between the two criteria may yield to a low-cardinality set of Pareto optimal solutions, as observed, for example, in [26] for the bi-objective quadratic assignment problem.

In the following, a graph characterization, of the MINSUM objective is introduced. Given a graph $H = (V, E)$, a *trail* is a sequence $T := (v_0, e_0, v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k)$, where $v_i \in V$,

$i = 0, \ldots, k$, $e_j \in E$, $j = 0, \ldots, k-1$, v_i and v_{i+1} are the endpoints of e_i, and the edges are all distinct. A *path* is a trail in which all nodes are distinct. An edge is *dominated* by a trail T if it either belongs to T or it is incident to a node of T. A *dominating trail* T_D is a trail such that each edge of H is dominated by T_D. Similarly, a *dominating trail set* Σ is a collection of edge-disjoint trails such that each edge of H is dominated by a trail in Σ. A *Minimum Dominating Trail Set* ($MDTS$) is a dominating trail set of minimum cardinality. Observe that a dominating trail set Σ on the bipartite graph $G = (V_1, V_2, E)$ completely specifies a batch schedule σ. In fact the dominated edges of a trail T in Σ can always be scheduled between two consecutive edges of T with no global changeovers (recall that, if two edges of G have one endpoint in common, we can switch from one to the other by means of a local changeover). If G is connected, the number of global changeovers is given by $|\Sigma|$. Therefore, a schedule without global changeovers exists if and only if there is a dominating trail on G. As a consequence, we can express MINSUM as the problem of dominating the edges of G with the minimum number of (edge-disjoint) trails, i.e., finding a $MDTS$. In Figure 1, the edge sequences σ_1 and σ_2 correspond, for example, to the dominating trail sets Σ_1 and Σ_2, respectively. Note that, $|\Sigma_1| = 2$ and $|\Sigma_2| = 3$.

Harary and Nash-Williams [18] link the problem of finding a $MDTS$ of H and the problem of finding the *Hamiltonian Completion Number* (HCN) of the line graph $L(H)$ of H. They show that $L(H)$ has a Hamiltonian path if and only if H has a dominating trail. As a consequence, if $HCN(L(H)) = k$ then the cardinality of $MDTS$ of H is $k + 1$.

3.1 Lower bounds and ideal solution

In this section, lower bounds for the MINSUM and MINMAX objectives are presented. Such bounds can be used to obtain an *ideal solution* for the problem [15]. **In [12], an algorithm for computing a lower bound on the cardinality of a $MDTS$, and, hence, for the MINSUM objective, is proposed. Given a general connected graph, the algorithm is based on a series of transformations that reduces the graph to a particular tree, with the property that the cardinality of a $MDTS$ on the tree (that can be computed in closed form) is a lower bound to the cardinality of a $MDTS$ on the original graph. Recalling that a bridge in a graph is an edge whose removal produces a disconnected graph, the tree is generated as follows. The nodes of the tree are obtained removing all the bridges from the graph and collapsing into single nodes all the remaining connected components, and the edges of the tree are the bridges in the graph.**

Given an instance of *bi-SC*, let $G = (V_1, V_2, E)$ be the associated bipartite graph. If G is connected, let Σ be a $MDTS$ on G. The number of global changeovers is given by $|\Sigma|$ and an optimal solution for MINSUM is $|E| + |\Sigma|$. In fact, given an edge sequence, a global changeover occurs and two setups are paid (one in each department) for each two consecutive edges with no endpoint in common, otherwise a local changeover occurs (one setup is paid). On the other hand, if the graph G is not connected, then it is possible to apply the above discussion to each connected component of G separately. The number of the global changeovers is, therefore, given by the sum of the global changeover of each connected component.

Let LB be the lower bound of $|\Sigma|$ computed as in [12]. A lower bound for the MINSUM objective is then $|E| + LB$, and it can be improved by

$$\max\{|E| + LB, |V_1| + |V_2|\}, \tag{3}$$

where the second term holds for sparse and disconnected graphs. In view of this observation, it follows that a lower bound on the maximum number of setups occurring in one department (MINMAX) is $\lceil (|E| + LB)/2 \rceil$. This bound can be improved by the formula

$$\max\{\lceil (|E| + LB)/2 \rceil, |V_1|, |V_2|\}. \quad (4)$$

Then, an ideal point (IP) for the bi-objective problem is the vector $(IP_{MINSUM}, IP_{MINMAX})$ where IP_{MINSUM} is given by Equation 3 and IP_{MINMAX} is given by Equation 4.

4 An algorithm for the $bi - SC$ problem

In this section we propose a metaheuristic approach developed to tackle the $bi - SC$ problem. A metaheuristic is an iterative solution procedure, combining subordinate heuristic tools into a more sophisticated framework. Such methods cover a large family of techniques, varying from rather simple to very sophisticated approaches [20, 14]. In particular, we consider the *Iterated Local Search* (ILS) [22] metaheuristic which is a conceptually very simple metaheuristic able to obtain state-of-the-art performance for several hard combinatorial problems [21, 7, 28].

This section is organized as follows. First, we introduce the neighborhoods for the MINSUM and MINMAX search spaces, respectively. Next, we describe two local search procedures for the two objectives and finally we combine these local searches in the Iterated Local Search approach (ILS) framework.

4.1 Neighborhoods for the MINSUM objective

In this section we describe the two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$, first introduced in [11] for the problem of finding a $MDTS$ on a general graph $H = (V, E)$. Given a dominating trail set Σ on H, a neighborhood structure $\mathcal{N}(\Sigma)$ specifies which solutions are "close" to Σ in some sense. The solutions $\Sigma' \in \mathcal{N}(\Sigma)$ are called neighbors of Σ and they can be reached from Σ. It is useful to introduce some notations about the structure of the dominating trail set Σ. A dominating trail set Σ has the property to partition the set of edges of H. In fact, each edge $e \in E$ either belongs to a trail of Σ or it is dominated by (at least) one of them. Without loss of generality, we assume each edge dominated exactly by one trail of Σ, i.e., in the case an edge is dominated by more than one trail we consider only one of them. In view of this assumption we associate to each trail $T \in \Sigma$ a sequence $\sigma(T)$ of edges in which subsequent edges have always one node in common, i.e., $\sigma(T)$ contains the edges of the trail T and the edges dominated by T. Note that, different edge sequences can be associated to a trail T. Each sequence $\sigma(T)$ has two *ending nodes* and two *ending edges*, referred to as *left* and *right* nodes and edges, respectively. Let v_r and e_r denote the right ending node and the right ending edge, respectively. By definition of $\sigma(T)$, the right ending edge $e_r = (v_r, v_{xr})$ is adjacent to an edge in the sequence $\sigma(T)$ sharing node v_{xr} with it. We call *right ending subsequence* $\sigma_r(T)$ of $\sigma(T)$ the maximal subsequence of $\sigma(T)$ such that all edges in $\sigma_r(T)$ contains node v_{xr}. Obviously, the right ending edge (v_r, v_{xr}) belongs to $\sigma_r(T)$. In Figure 2, the trail $T = \{(3,4),(4,6)\}$ and the (possible) sequence $\sigma(T) = \{(1,3),(2,3),(3,4),(4,5),(4,6),(6,7)\}$ are reported. The right ending node and the right ending edge are respectively 7 and $(7,6)$ and node $v_{xr} = 6$. Then the *right ending subsequence* is $\sigma_r(T) = \{(4,6),(6,7,)\}$. Given an ending subsequence $\sigma_r(T) = \{e_r, \ldots, e_y\}$, where e_r is the right ending edge and e_y the other terminal edge of

the subsequence, we call *right terminal subset* the set of edges containing all edges in $\sigma_r(T)$ except the terminal edge e_y, i.e., $E_r(T) = \sigma_r(T) \setminus \{e_y\}$. $E_r(T)$ may contains a single edge. In Figure 2, the right terminal subset of $\sigma_r(T) = \{(4,6),(6,7)\}$ contains the single edge $(4,6)$. Any permutation of the edges in $E_r(T)$, if more than one edge exists, yields a new sequence $\sigma'(T)$ still representing the trail T, i.e., such that two subsequent edges in $\sigma'(T)$ have a node in common. Similar definitions can be given when the left ending edge and the left ending node are considered in $\sigma(T)$, leading to the definitions of *left ending subsequence* $\sigma_l(T)$ and *left terminal subset* $E_l(T)$, respectively. Hence, in the example of Figure 2, the left ending node and the left ending edge are respectively 3 and $(1,3)$ and node $v_{xl} = 3$. The left ending subsequence is $\sigma_l(T) = \{(1,3),(2,3),(3,4)\}, e_l = (1,3)$, and the left terminal subset is $E_l(T) = \{(1,3),(2,3)\}$. Note that, by permuting the edges in $E_l(T)$ a new sequence $\sigma'(T)$ still representing the trail T is obtained, i.e., such that two subsequent edges in $\sigma'(T)$ have a node in common.
Therefore to each $T \in \Sigma$ we associate a sequence $\sigma(T)$ and a *terminal subset* $S(T) = E_r(T) \cup E_l(T)$. As explained in the following, edges in $S(T)$ can allow to link up the sequence associated to another trail in Σ.

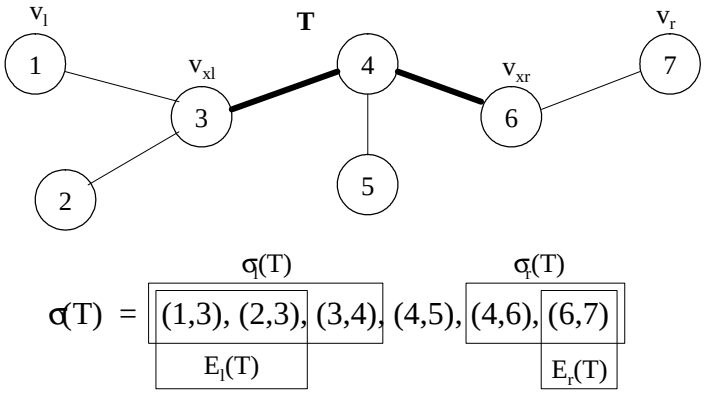

Figure 2: Ending subsequences and terminal subsets.

Given a solution Σ, we consider the neighborhood $\mathcal{N}_1(\Sigma)$ obtained by merging two trails having two adjacent edges in their terminal subsets, i.e., sharing a common node. More formally, given two trails T_1 and T_2, let $e_h \in S(T_1)$ and $e_k \in S(T_2)$ be two edges sharing a node v_i. By definition, the two terminal subsets $S(T_1)$ and $S(T_2)$ can be sequenced in such a way that the correspondent ending edges are e_k and e_h respectively. Hence, the trails T_1 and T_2 can be merged in a single trail leading to a decrease of the MINSUM objective function. In Figure 3(a), a graph H is represented showing a dominating trail set Σ, containing four trails (the bold edges). Moreover for each trail T_i the set $S(T_i)$ is indicated. Note that a node shared by the two terminal subsets $S(T_2)$ and $S(T_4)$ exists. Hence a move in the neighborhood $\mathcal{N}_1$ is possible. In Figure 3(b) the same graph is shown after the move has been done in $\mathcal{N}_1$, where the trails T_2 and T_4 are now merged.

Given a solution Σ, the neighborhood $\mathcal{N}_2(\Sigma)$ is obtained by splitting a trail into two trails and merging them with two different pre-existent trails. More precisely, let $\Sigma = \{T_0, T_1, \ldots, T_q\}$ and let us suppose that two consecutive edges e_i and e_{i+1} in $\sigma(T_0)$ share a node with an element of $S(T_1)$ and $S(T_2)$, respectively. Then it is possible to split trail T_0 into two trails and merge them with trails T_1 and T_2. Clearly, after applying a move in the neighborhood $\mathcal{N}_2$, the MINSUM objective function decreases by one, since three trails are rearranged in two trails. In Figure 3(c), an example of a move in $\mathcal{N}_2$ applied to the graph in Figure 3(b) is shown.

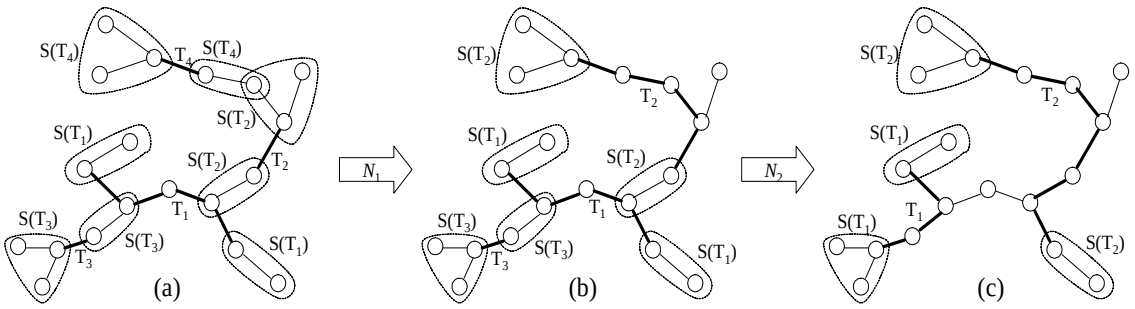

Figure 3: A move in $\mathcal{N}_1$ and $\mathcal{N}_2$.

In this case, trail T_1 is split and merged with T_3 and T_2. Now we discuss neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ in terms of scheduling terminology. Recall that a node of the bipartite graph represents an attribute on a department and an edge in the graph corresponds to a batch to be produced. A trail T_i is represented by a sequence of edges (batches) $\sigma(T_i)$ to be processed by the manufacturing system. A trail set $\Sigma = \{T_0, T_1, \ldots, T_q\}$ can be represented by any sequence of the trail sequences $\sigma(T_i)$, i.e., $\sigma(\Sigma) = \{\sigma(T_0), \sigma(T_1), \ldots, \sigma(T_q)\}$. Hence, a move in $\mathcal{N}_1$ corresponds to the insertion of a subsequence $\sigma(T_i)$ into a specified position in $\sigma(\Sigma)$, whereas a move in $\mathcal{N}_2$ performs two subsequence insertion operations.

4.2 Neighborhood for the MINMAX objective

In this section, a neighborhood for the MINMAX objective called $\mathcal{N}_B(\Sigma)$ is introduced. Given a trail set Σ, the neighborhood $\mathcal{N}_B(\Sigma)$ is an insertion neighborhood in which each neighbor of the sequence $\sigma(\Sigma)$ is simply obtained by moving a single edge e_i from its position in the sequence to another position. In other words, given $\sigma(\Sigma) = \{e_1, e_2, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_j, e_{j+1}, \ldots, e_{|E|}\}$, then a neighbor Σ' in $\mathcal{N}_B(\Sigma)$ is the sequence $\sigma(\Sigma') = \{e_1, e_2, \ldots, e_{i-1}, e_{i+1}, \ldots, e_j, e_i, e_{j+1}, \ldots, e_{|E|}\}$. Such insertion neighborhood, in general, admits $O(|E|^2)$ possible moves, since each edge can be inserted in $O(|E|)$ positions. However, an efficient neighborhood pruning scheme is proposed in order to reduce the computational effort of exploring $\mathcal{N}_B(\Sigma)$. In particular, we consider only those edges that would lead to improvements of the MINMAX objective function.

If the edge $e = (i, j)$ is dominated by a trail passing in i (j), where i (j) represents an attribute related to department D_1 (D_2), then a local changeover is paid by the department D_2 (D_1). Given a trail set Σ, we restrict our attention to the edges $e = (i, j)$ not laying on a trail and that can be dominated by both their nodes i and j. More formally, we only consider in $\mathcal{N}_B$ those edges $e = (i, j)$ such that $e \notin T_p, \forall T_p \in \Sigma$, $i \in T_h$ and $j \in T_k$, where T_h and T_k are trails of Σ, possibly $h = k$. For example, if e is currently dominated by a trail T_h passing in i and another trail T_k passing in j exists, then it is possible to dominate e by the trail T_k passing in j. Hence, e may be moved from its current position in the sequence to another position, in such a way that it is dominated by the trail T_k.

Note that a move in $\mathcal{N}_B$ does not affect the MINSUM objective function, since moving edge e splits no trail. In fact, removing e from its current position in $\sigma(T_h)$ does not split trail T_h since edge e is only dominated by trail T_h. On the other hand, it is possible to suitably insert e in $\sigma(T_k)$ without affecting the cardinality of the set Σ.

2-Neighborhood Descent (2ND)
begin
  $k := 1$
  while $k < 3$ **do**
    if $k = 1$ search for a move in $\mathcal{N}_1$
    else search for a move in $\mathcal{N}_2$
    if a move is found **then** apply the move
    else $k := k + 1$
  end
end

Figure 4: Algorithmic scheme of 2ND.

4.3 A Local Search algorithm for the MINSUM objective

Based on the definition of the neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ for the MINSUM objective, we describe a fast local search procedure first introduced in [11]. In particular we implement a 2-Neighborhood Descent (2ND) algorithm, see Figure 4. The 2ND procedure falls in the well known Variable Neighborhood Descent (VND) framework [17], when only two neighborhoods are defined.

The heuristic performs a first-improvement exploration of the neighborhood $\mathcal{N}_1$ until a local minimum is reached. Once a local minimum in $\mathcal{N}_1$ is reached, moves are performed using a first-improvement exploration of the neighborhood $\mathcal{N}_2$ until a new local minimum (in $\mathcal{N}_2$) is reached. The 2-Neighborhood Descent procedure stops when no moves in $\mathcal{N}_2$ are possible. Since the 2ND accepts only strictly improving moves this algorithm converges towards a local minimum in a small number of iterations. Recall that every improving move reduces MINSUM objective by one. Once a local minimum in $\mathcal{N}_1$ is found the successive local search in $\mathcal{N}_2$ produces a solution which is a local minimum for both the neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$. In fact, the trail splitting move in $\mathcal{N}_2$ does not produce any new terminal subset, since no new ending subsequences are generated. Therefore no move in $\mathcal{N}_1$ can be performed.

Neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ can be visited in the worst case $O(|E|)$ times. At this aim, a particular data structure is used, in which for each node we store if the node belongs to a terminal subset and its position in the sequence. Whether an edge belongs to a terminal subset of a trail can be checked in constant time. Once a terminal subset edge (i, j) is found, the algorithm verifies if a terminal subset containing either i or j has already been found. If i (j) does not belong to a terminal subset then the data structure is updated to store information on i (j), otherwise a move in $\mathcal{N}_1$ has been found. In fact, two terminal subset sharing a common node have been identified. Since this operation can be performed in constant time at most $O(|E|)$ operations are required to explore neighborhood $\mathcal{N}_1$.

Neighborhood $\mathcal{N}_2$ can be explored in $O(|E|)$ too. In fact, a move in $\mathcal{N}_2$ is found when two consecutive edges sharing a node in any terminal subset are detected. This can be done in $O(|E|)$ operations using the previously described data structure.

4.4 A Local Search algorithm for the MINMAX objective

Now, we introduce a fast local search procedure called LS_{MINMAX}, based on the neighborhood $\mathcal{N}_B(\Sigma)$ described in Section 4.2. The local search performs a first-improvement exploration strategy of the neighborhood $\mathcal{N}_B(\Sigma)$ until a local minimum is reached.

Given a trail set Σ and an edge e that does not belong to any trail in Σ, a move in $\mathcal{N}_B(\Sigma)$ corresponds to change the position of e in the sequence $\sigma(\Sigma)$. Note that, by definition of $\mathcal{N}_B(\Sigma)$, this move does not modify Σ. In other words a move in $\mathcal{N}_B(\Sigma)$ does not affect the MINSUM objective, while it may modify the MINMAX objective. In the local search LS_{MINMAX}, a move in $\mathcal{N}_B$ is performed only if it leads to a more balanced solution, i.e., if it causes a decrease of the MINMAX objective. In practice, for each edge, the procedure checks if moving it leads to an improvement of the MINMAX objective. Once a profitable edge is detected the algorithm looks for a suitable position in the sequence. Checking whether an edge belongs to $\mathcal{N}_B(\Sigma)$ and where it must be inserted in the sequence can be done in constant time, by accessing a suitable data structure, in which for each node it is stored the trail containing it (if any) and its position in the sequence σ, and for each edge it is stored its position in the sequence σ and if it belongs or not to any trail of Σ. Given an edge (i, j) it can be checked in constant time if it does not lie on a trail of Σ. In this case $(i, j) \in \mathcal{N}_B(\Sigma)$ if the two nodes i and j belong to a trail. Hence, a move in $\mathcal{N}_B(\Sigma)$ can be performed in costant time knowing the position in the sequence σ of edge (i, j) and of nodes i and j. This data structure can be built in $O(|E|)$ time by simply scanning the trail set Σ and the sequence σ. Hence, from a computational point of view, the neighborhood $\mathcal{N}_B(\Sigma)$ can be explored in $O(|E|)$ time. Moreover, note that the order in which the moves are performed is independent. Hence, LS_{MINMAX} requires $O(|E|)$ time in total.

4.5 The Iterated Local Search algorithm

The basic scheme of an Iterated Local Search is given in Figure 5. The main idea behind the ILS is to perturbate the incumbent solution and to apply a new local search starting from the perturbated solution. When a new local minimum is obtained, an evaluation criterion decides whether to accept the new solution as a new incumbent solution or to discard it.

Due to the peculiar properties of the objective functions MINSUM and MINMAX of the *bi-SC* problem it is possible to extend the ILS framework to tackle our bi-objective problem. As already observed, the two objective functions (1) and (2) are correlated, since they both depends on the setup number in the departments D_1 and D_2. As a consequence reducing the MINSUM objective function may also lead to a reduction of the MINMAX objective. Moreover, as shown in Section 4.2, the local search for the MINMAX objective does not modify the solution value of the MINSUM objective, i.e., it does not vary the cardinality of the dominating trail set. These observations justify to design a procedure in which the MINSUM and MINMAX local searches are performed in cascade. Moreover, since our aim is to find a set of non-dominated solutions, i.e., the *solution front*, we propose a modified acceptance criterion to store in an archive the Pareto Set, that we call *Archive Better*.

In the remaining of this section we describe in details all the parts of the proposed ILS algorithm.

- As *Initial Solution* we consider the solution obtained by two simple greedy algorithms. The Maximum Degree Greedy algorithm (MaxDG) and the Minimum Degree Greedy algorithm (MinDG). At each step, the algorithm MaxDG (MinDG) identifies the node

Iterated Local Search (ILS)
Given an initial solution
begin
  Local Search
  while Stopping Criterion = false **do**
    Perturbation
    Local Search
    Acceptance Criterion
  end
end

Figure 5: Algorithmic scheme of ILS.

i with maximum (minimum) degree $d(i)$, and a new trail T is generated. The new trail is composed of a single edge adjacent to the selected node i, and it is added to Σ. In these algorithms only the edges incident to node i are considered dominated by the new trail T. The dominated edges are "removed" from the graph and the node degrees are updated correspondingly. This process is repeated until all the edges of the graph are dominated.

- The *Perturbation phase* applies random moves to the incumbent solution. A perturbation move splits randomly a trail. A perturbation is characterized by a parameter $p \in (0, 1)$ called *perturbation strength* which indicates the maximum number of random moves that can be applied to the incumbent solution. The actual number of perturbating moves is randomly drawn in the $[1, p * |E|]$ interval.

- In the *Local Search phase*, we perform the two local search procedures previously introduced. As mentioned before, the solution found by the MINMAX local search does not affect the MINSUM solution value. Hence, first, the 2ND (Figure 4) is applied in order to improve the MINSUM objective, and then, starting from the local minimum of MINSUM, the MINMAX local search is performed, thus reducing the MINMAX objective, without affecting the MINSUM value.

- The *Acceptance Criterion* evaluates the new solutions generated by the local search phase and decides whether to discard or to store them. Since our aim is to find a good approximation of the Pareto front, we propose the *Archive Better* acceptance criterion able to store the current solution front, i.e., the set of non-dominated solutions. In particular, the algorithm collects in an *archive* all the not dominated solutions found during the search process. When a new incumbent solution is obtained, it is compared with the solutions stored in the archive. If the incumbent solution is non-dominated by any archived solution it is added to the archive. Moreover all the solutions in the archive dominated by the incumbent solution are removed from it, i.e., the archive is constantly updated to store the Pareto front found so far. The subsequent iteration of the ILS starts from the last archived solution.

- The *Stopping Criterion* of the ILS algorithm is established in a given number of iterations without adding a non-dominated solution to the archive. Moreover, if during the search

process the ideal point is reached the algorithm stops. In this case the optimal Pareto front contains only a single point.

Besides ILS, in the next section, we also evaluate a straightforward local search algorithm (called LS) consisting of the Local Search phase of the ILS. Clearly, this procedure will obtain only a single solution since it is a simple local search and it is not modified to deal with a bi-objective problem.

5 Computational experiments

In this section we describe the experiments carried out to evaluate the behavior of the proposed algorithms.

5.1 Instance description

The ILS and the LS algorithms have been tested on a set of real-life instances and two sets of randomly generated bipartite graphs presented first in [2].

The set of real world instances (Set RW) consists of 32 instances arising in a kitchen furniture manufacturer production line. These instances have an high density of the graph ($d \geq 50\%$), and the resulting bipartite graphs are not balanced, in fact, in Set RW department D_1 has about 30 product classes (shapes), while department D_2 has about 15 different classes (colors).

The first set of randomly generated instances, denoted as Set 1, consists of balanced bipartite graphs $G = (V_1, V_2, E)$, i.e., graphs in which the sets V_1 and V_2 have the same cardinality. The instances in this set are generated, varying the cardinality of the node sets $n = |V_1| = |V_2|$ from 10 to 100 and the graph density from 5% to 40%. The values 10, 30, 60, 80, 100 have been considered for n, and 5, 10, 20, 30, 40% for the density d. For each pair (n, d) a set of 20 connected instances has been generated.

The second set of instances (Set 2) consists of randomly generated bipartite graphs in which a dominating trail *does* exist. In these instances $N = |V_1| + |V_2|$ varies from 10 to 100 and 90 instances for each value of N are generated. Note that these graphs are not balanced, i.e., in general $|V_1| \neq |V_2|$.

In a preliminary test phase we tuned the proposed algorithms. We tested the influence of the initial solution and the perturbation strength p. The best configuration found for the ILS algorithm is with maximum perturbation strength $p = 0.15$, the Archive Better acceptance criterion and MinDG heuristic as initial solution. All the experiments have been performed on a 1.5 GHz Pentium M laptop equipped with 632 MB ram, and the algorithms are coded in standard C language.

5.2 Performance measures

Appropriate metrics must be selected in order to evaluate the behavior of an algorithm. The literature offers different metrics to measure the performance of algorithms for multicriteria combinatorial optimization problems. Nevertheless, no single metric is able to entirely point out the total algorithmic performance. In what follows, the metrics adopted in this study are reported. Clearly, these metrics should not be considered as a complete list of all the possible

metrics. For instance, in our computational experiments, we do not consider particular temporal metrics, limiting our analysis only to the computation times required by the algorithm. Although presented in a bi-objective context, the considered metrics can be applied also in the case with an higher number of objectives.

More in details, we measure how far the non-dominated solutions obtained by the algorithm, i.e. the solution front (SF), are from the ideal point (IP) known on the basis of theoretical bounds on the objectives. We use also the concept of nadir point. A nadir point (NP) is defined as a point characterized by worst values for each objectives. In particular, in the problem under study, the nadir point is trivially determined as the point with $NP_{MINSUM} = 2 * |E|$, and $NP_{MINMAX} = |E|$.
We consider a first set of measures of performance D_1, D_2 and D_∞ based on the l_1, l_2 and l_∞ norms, respectively. Hence, we calculate

$$D_p = \frac{1}{|\mathrm{SF}|}\left(\sum_{i=1}^{2}\left(\frac{\mathrm{SF_i(j)} - \mathrm{IP_i}}{\mathrm{NP_i} - \mathrm{IP_i}}\right)^p\right)^{\frac{1}{p}}, \tag{5}$$

where $SF_i(j)$ is the i-th component of the j-th vector contained in SF. Note that as nadir and ideal points limit the interval of values assumed by each objective, they are used to normalize the measures D_p.

We adopt other two measures of performance. The first measure is given by the number of solutions that can *potentially dominate* the solutions in SF, from now on we refer to this metric as *potentially dominating solutions* (PDS). PDS is determined counting the potential solution points between the IP and the SF able to strictly dominate the solutions in SF. Note that multiple potential solutions mapping the same point are considered once. Moreover, a potential solution point may not correspond to a real solution of the problem.

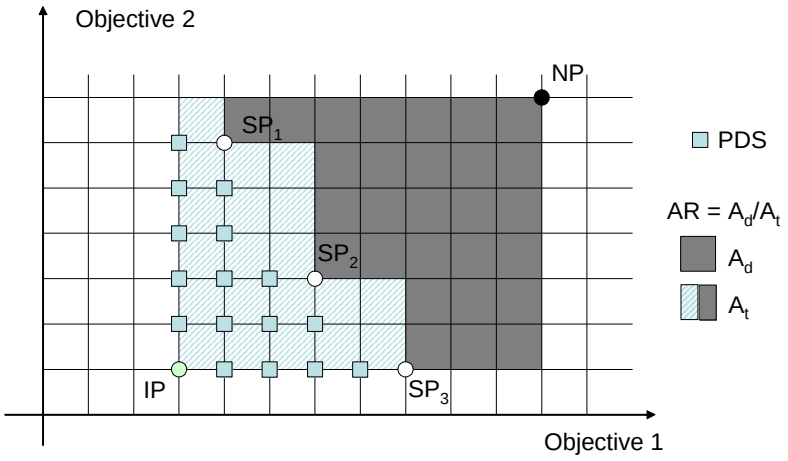

Figure 6: The potentially dominating solutions (PDS) and the area ratio (AR).

The second measure is given by an area ratio. This metric requires to know an ideal point (IP) and a nadir point (NP) for the problem. The ideal point and the nadir point can be viewed as vertices of an regular polytope defining an hypervolume A_t, i.e. the total area, in the space of the objectives (a rectangle in the two objective case). The area ratio (AR) performance measure is defined as $AR = \frac{A_d}{A_t}$, where A_d indicates the *dominated* area between the nadir point NP and the solution front SF, as proposed in [16]. In Figure 6, the ideal point

Time	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
<0.01	0.0001	0.0001	0.0001	0.0313	0.9999	1.00

Table 1: Set RW: LS's results on the real world instances.

(IP), the nadir point (NP), the solution front ($SF = \{SP_1, SP_2, SP_3\}$) and the potentially dominating solutions (PDS) are shown. The dominated area A_d (the dark grey area) and the total area A_t (the light grey area plus the dark grey area) are also represented in the figure.

5.3 Performance analysis

In Tables 1–3 we report on the performance of LS algorithm on the three test sets; whereas Tables from 4 to 6 reports on the results of the ILS algorithm. Finally in Table 7 we show a comparison between the two proposed approach. In each table, the results obtained by the algorithms are presented taking into account the performance measures previously described. In particular, the computation time in seconds (Time), the iterations required by the algorithm (Iter), the average distance value of the SF from the IP for l_1, l_2 and l_∞ norms (D_1, D_2, D_∞) are shown, respectively. Moreover, we also report on the number of potentially dominating solutions (PDS), the area ratio (AR) and the cardinality of the solution front ($|SF|$). In Tables 2 and 5 the first three columns report on the cardinality of the node sets ($n = |V_1| = |V_2|$), the graph density d and the average number of edges in the instance set ($|E|$), respectively. On the other hand Tables 3 and 6 report in the first column the total cardinality of the node set ($N = |V_1| + |V_2|$).

In Tables 1–3, the results of the LS algorithm are shown. On Set RW, the LS algorithm is able to solve to optimality, i.e., reaching the ideal point, almost all the industrial instances. Regarding Set 1 (Table 2) we observe that the PDS index is often smaller than 1, thus it shows that the LS algorithm is able to provide good quality solutions close to the ideal point. In particular, for the dense instances of Set 1 the algorithm is always able to reach the ideal point, and consequently $AR = 1$ and $PDS = 0$. The covered area is always greater than 80%, and the number of solutions that potentially dominate the solution found is rarely greater than 5. Considering Set 2 (Table 3) as the size of the instances grows, we observe a reduction of the average distances (D_1, D_2, D_∞) and an increase of the area ratio (from 89% to 95%), due to the normalization effect of these metrics. On the other hand, as the size of the instances grows the PDS index grows (from 1 to 14) due to the fact the the PDS is an absolute measure and no normalization is taken into account. Moreover we observe that the LS algorithm performs better on Set 1 than on Set 2. As expected, $|SF| = 1.00$, hence the solution front always contains only one solution. The computation times required by the LS are negligible, since they never exceed 0.01 second. Therefore the LS algorithm is able to attain reasonable quality solutions within very strict computation times.

In Tables 4–6, the results of the ILS are presented. Since the perturbation phase of the ILS algorithm is random, the results of the ILS are average values related to 10 runs of the algorithm. Also for the ILS, in Set 1 (Table 5), the sparse instances result more difficult to solve when compared with more dense instances. In fact, the number of iterations required and PDS values are greater, and the area ratio is smaller, when compared to the same values for more dense instances. Regarding the computation times, we observe that rarely the algorithm requires more than 1 second of CPU time. The area ratio AR is always greater than 98%, and frequently the optimal Pareto front, i.e., the ideal point, is reached. The PDS value never

n	d	$\|E\|$	Time	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
10	0.1	21.0	<0.01	0.1100	0.1698	0.1282	4.1000	0.8386	1.00
10	0.2	24.9	<0.01	0.1115	0.1645	0.1266	4.8000	0.8441	1.00
10	0.3	29.6	<0.01	0.0493	0.0796	0.0590	2.7500	0.9230	1.00
10	0.4	37.1	<0.01	0.0203	0.0302	0.0235	1.1500	0.9704	1.00
30	0.1	97.9	<0.01	0.0166	0.0283	0.0211	3.2500	0.9720	1.00
30	0.2	177.1	<0.01	0.0011	0.0014	0.0012	0.2500	0.9986	1.00
30	0.3	266.4	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
30	0.4	355.9	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
60	0.05	197.1	<0.01	0.0165	0.0303	0.0216	9.0000	0.9700	1.00
60	0.1	360.5	<0.01	0.0021	0.0034	0.0025	1.4000	0.9966	1.00
60	0.2	713.7	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
60	0.3	1065.3	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
60	0.4	1418.0	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
80	0.05	324.0	<0.01	0.0076	0.0134	0.0097	5.9500	0.9867	1.00
80	0.1	641.4	<0.01	0.0008	0.0011	0.0009	0.7500	0.9989	1.00
80	0.2	1272.6	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
80	0.3	1900.0	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
80	0.4	2529.1	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
100	0.05	499.5	<0.01	0.0035	0.0053	0.0041	3.3500	0.9947	1.00
100	0.1	1000.5	<0.01	0.0002	0.0002	0.0002	0.2000	0.9998	1.00
100	0.2	1990.6	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
100	0.3	2981.8	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00
100	0.4	3959.8	<0.01	0.0000	0.0000	0.0000	0.0000	1.0000	1.00

Table 2: Set 1: LS's results on bipartite instances.

N	Time	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
10	<0.01	0.0878	0.1089	0.0921	1.1461	0.8956	1.00
20	<0.01	0.0566	0.0784	0.0623	2.0556	0.9248	1.00
30	<0.01	0.0350	0.0489	0.0392	2.3333	0.9531	1.00
40	<0.01	0.0346	0.0501	0.0394	3.4111	0.9521	1.00
50	<0.01	0.0292	0.0454	0.0344	4.3667	0.9565	1.00
60	<0.01	0.0276	0.0438	0.0326	5.5333	0.9581	1.00
70	<0.01	0.0290	0.0469	0.0350	8.4111	0.9556	1.00
80	<0.01	0.3778	0.0434	0.0327	9.8667	0.9589	1.00
90	<0.01	0.0271	0.0439	0.0325	12.3778	0.9585	1.00
100	<0.01	0.0266	0.0434	0.0322	14.2778	0.9589	1.00

Table 3: Set 2: LS's results on instances with dominating trail.

Time	Iter	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
<0.01	0.28	0.0000	0.0000	0.0000	0.00	1.0000	1.00

Table 4: Set RW: ILS's results on the real world instances.

n	d	$\|E\|$	Time	Iter	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
10	0.1	21.0	<0.01	1016.30	0.0140	0.0169	0.0147	0.28	0.9834	1.00
10	0.2	24.9	<0.01	337.80	0.0047	0.0069	0.0052	0.15	0.9933	1.00
10	0.3	29.6	<0.01	25.86	0.0000	0.0000	0.0000	0.00	1.0000	1.00
10	0.4	37.1	<0.01	26.44	0.0000	0.0000	0.0000	0.00	1.0000	1.00
30	0.1	97.9	0.02	514.03	0.0007	0.0009	0.0007	0.08	0.9991	1.00
30	0.2	177.1	<0.01	11.65	0.0000	0.0000	0.0000	0.00	1.0000	1.00
30	0.3	266.4	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
30	0.4	355.9	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
60	0.05	197.1	0.44	1965.64	0.0020	0.0029	0.0022	0.56	0.9972	1.00
60	0.1	360.5	0.06	50.12	0.0000	0.0000	0.0000	0.00	1.0000	1.00
60	0.2	713.7	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
60	0.3	1065.3	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
60	0.4	1418.0	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
80	0.05	324.0	0.98	1145.01	0.0005	0.0007	0.0006	0.22	0.9993	1.00
80	0.1	641.4	0.54	87.49	0.0000	0.0000	0.0000	0.00	1.0000	1.00
80	0.2	1272.6	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
80	0.3	1900.0	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
80	0.4	2529.1	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
100	0.05	499.5	1.54	462.26	0.0000	0.0000	0.0000	0.01	1.0000	1.00
100	0.1	1000.5	0.07	6.37	0.0000	0.0000	0.0000	0.00	1.0000	1.00
100	0.2	1990.6	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
100	0.3	2981.8	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00
100	0.4	3959.8	<0.01	0.00	0.0000	0.0000	0.0000	0.00	1.0000	1.00

Table 5: Set 1: ILS's results on bipartite instances.

exceeds 1, showing that the algorithm, on average, either attains the ideal point or a solution close to it. Also the instances of Set 2 (Table 6) are easily solved by the ILS algorithm, since only a small number of iterations is required to solve the instance to the optimality. Besides we observe only one case with $|SF| > 1$, that is for one instance more than a single solution is stored in the archive.

The sparse instances are, in general, harder to solve compared with dense instances. When comparing instances having same density but different size, we observe that the larger instances seem to be easier to solve. Consider the following cases. Given a sparse and small instance with 20 nodes and 21 edges, a node has on average only one edge adjacent to it. On the other hand, given a large and dense instance, say with 200 nodes and 4000 edges each node has 20 adjacent edges. Hence a dense instance, even if it is larger, is easier to solve because the algorithm has a lot of degree of freedom while building a solution. Finally we observe that the differences between the distance values (D_1, D_2, D_∞) are always negligible.

Table 7 summarize the behavior of the two proposed algorithms on the three considered test sets. It turns out that both the algorithms are fast enough to solve industrial scheduling problems in real time, and that the ILS is able to improve the solution quality found by the basic LS requiring a small additional computation time.

Moreover, we compare our algorithms with the genetic algorithms independently proposed in [25, 24]. In [25] different versions of this problem with general setup costs are presented and a multi-objective genetic approach is proposed. ILS and LS outperform the genetic algorithms

N	Time	Iter	D_1	D_2	D_∞	PDS	AR	$\|SF\|$
10	<0.01	4.33	0.0000	0.0000	0.0000	0.00	1.0000	1.00
20	<0.01	66,55	0.0009	0.0009	0.0009	0.01	0.9983	1.01
30	<0.01	8.08	0.0000	0.0000	0.0000	0.00	1.0000	1.00
40	<0.01	14.25	0.0000	0.0000	0.0000	0.00	1.0000	1.00
50	<0.01	17.21	0.0000	0.0000	0.0000	0.00	1.0000	1.00
60	<0.01	25.48	0.0000	0.0000	0.0000	0.00	1.0000	1.00
70	<0.01	24.75	0.0000	0.0000	0.0000	0.00	1.0000	1.00
80	<0.01	34.53	0.0000	0.0000	0.0000	0.00	1.0000	1.00
90	<0.01	38.44	0.0000	0.0000	0.0000	0.00	1.0000	1.00
100	0.02	79.98	0.0000	0.0000	0.0000	0.00	1.0000	1.00

Table 6: Set 2: ILS's results on instances with dominating trail.

Set	Instances	LS			ILS		
		Time	PDS	AR	Time	PDS	AR
RW	32	<0.01	0.0313	0.9999	<0.01	0.0000	1.0000
1	460	<0.01	1.6065	0.9987	0.16	0.5652	0.9987
2	900	<0.01	6.3779	0.9472	<0.01	0.0012	0.9998

Table 7: Comparisons between LS and ILS.

both in terms of quality and computation time. In fact, the genetic algorithms needs several minutes to obtain a sub-optimal solution for a real world instances from Set RW, whereas the proposed algorithms require less than 0.01 seconds to reach the ideal solution. Recently, Mansouri [24] developed another genetic algorithm (MOGA) for the $bi-SC$ problem. In the computational experiments carried out on one real-world instance (Set RW) and on randomly generated instances up to 80 nodes for each department, MOGA is able to achieve good quality solutions in few minutes. In particular, when evaluating the performance of LS, ILS and MOGA algorithms on comparable instances (similar to the instances of Set 1 with $n = 30$ and $n = 80$), LS and ILS are able to solve them to the optimality requiring less than 0.5 seconds, whereas MOGA is not able to reach the optimal solution within two minutes of computation.

6 Conclusions

In this paper, we considered a scheduling problem arising in a two–stage industrial plant with no intermediate buffer. The problem consists in finding a batch common sequence optimizing both the total number of setups and the maximum number of setups paid by a single stage. A generalization of the ILS metaheuristic to the bi-objective problem is proposed for this problem, and it is tested on a wide set of instances. Due to the particular problem structure and to the correlation between the two objective functions the proposed approach results effective. In fact, the computational results show the effectiveness with respect to both the quality of the solutions and the computation times. In particular, over the 1300 considered instances, the ILS is able to attain solutions extremely close to the theoretical bound. Future research directions include the study of improved algorithmic schemes and the extension of the approach to problems with a more general cost structure.

References

[1] Agnetis, A., Detti, P., Meloni, C., (2003), *Process selection and sequencing in a two-agents production system*, 4OR, 1 (2), 103–119.

[2] Agnetis, A., Detti, P., Meloni, C., Pacciarelli, D., (2001), *Set–up coordination between two stages of a supply chain*, Annals of Operations Research, 107, 15–32.

[3] Al-Fawzan, M.A., Al-Sultan, K.S., (2002), *A tabu search based algorithm for minimizing the number of tool switches on a flexible machine*, Computers & Industrial Engineering, 44, 35–47.

[4] Al–Haboubi, M.H., Selim, S.Z., (1993), *A sequencing problem in the weaving industry*, European Journal of Operational Research, 66, 65–71.

[5] Allahverdi, A., (2003), *The two- and m-machine flowshop scheduling problems with bicriteria of makespan and mean flowtime*, European Journal of Operational Research, 147, 373–396.

[6] Allahverdi, A., Gupta J.N.D., Aldowaisan, T., (1999), *A review of scheduling research involving setup considerations*, Omega, 27, 219–239.

[7] Balas, E., Vazacopoulos, A., (1998), *Guided local search with shifting bottleneck for job shop scheduling*, Management Science, 44 (2), 262–275.

[8] Crama, Y., Oerlemans, A.G., Spieksma F.C.R., (1996), *Production Planning in Automated Manufacturing*, Springer–Verlag, Berlin.

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

[10] Detti, P., Meloni, C., (2001), *Part type selection and batch sequencing in a two-stage manufacturing system*, Proceedings of 16th International Conference on Production Research, Praha.

[11] Detti, P., Meloni, C., Pranzo, M., (2003), *Local Search Algorithms for the Minimum Cardinality Dominating Trail Set of a Graph*, Technical report RT-DIA-84-2003, Dipartimento di Informatica e Automazione, Università Roma Tre, Roma, Italy.

[12] Detti, P., Meloni, C., Pranzo, M., (2004), *Simple bounds for the minimum cardinality dominating trail set problem*, Technical report RT-DIA-87-2004, Dipartimento di Informatica e Automazione, Università Roma Tre, Roma, Italy.

[13] Esswein, C., Billaut, J.C., Strusevich, V.A., (2005), *Two-machine shop scheduling: Compromise between flexibility and makespan value*, European Journal of Operational Research, 167, 796–809.

[14] Glover, F., Kochenberger, G., (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers.

[15] Ehrgott, M., (2000), *Multicriteria Optimization*, Lecture Notes in Economics and Mathematical Systems, Springer–Verlag.

[16] Fleischer, M., (2003), *The measure of Pareto optima: Applications to multi-objective metaheuristics*, Lecture Notes in Computer Science, 2632, 519–533.

[17] Hansen, P., Mladenović, N., (2001), *Variable neighborhood search: Principles and applications*, European Journal of Operational Research, 130, 449–467.

[18] Harary, F., Nash-Williams, C.St.J.A., (1965), *On Eulerian and Hamiltonian graphs and line-graphs*, Canadian Mathematics Bulletin, 8, 701–709.

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

[20] Hoos, H.H., Stützle, T., (2004), *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann Publishers.

[21] Johnson, D.S., McGeoch, L.A., (1997) *The travelling salesman problem: a case study in local optimization*, Local Search in Combinatorial Optimization, John Wiley and Sons, New York, 215–310.

[22] Lourenço, H.R.D., Martin, O., Stützle, T., (2002), *Iterated Local Search* In Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics, Kluwer, 321–353.

[23] McIntosh, R.I., Culley, S.J., Mileham, A.R., Owen , G.W., (2001), *Changeover improvement: A maintenance perspective*, International Journal of Production Economics, 73, 153–163.

[24] Mansouri, S.A., (2005), *Coordination of set-ups between two stages of a supply chain using multi-objective genetic algorithms*, International Journal of Production Research, 43, (15), 3163–3180.

[25] Meloni, C., Naso, D., Turchiano, B., (2005), *Setup coordination between two stages of a production system: a multi-objective evolutionary approach*, Annals of Operations Research, (to appear).

[26] Paquete, L., Stützle, T., (2006), *A Study of Stochastic Local Search Algorithms for the Biobjective QAP with Correlated Flow Matrices*, European Journal of Operational Research, 169, (3), 943–959.

[27] Ríos–Mercado, R.Z., Bard, J.F., (1998), *Heuristics for the flow line problem with setup costs*, European Journal of Operational Research, 110, 76-98.

[28] Smyth, K., Hoos, H.H., Stützle, T., (2003), *Iterated Robust Tabu Search for MAX-SAT*, Lecture Notes in Computer Science, 2671, 129–144.

[29] Spina, R., Galantucci, L.M., Dassisti, M., (2003), *A hybrid approach to the single line scheduling problem with multiple products and sequence-dependent time*, Computers & Industrial Engineering, 45, 573–583.

[30] T'kindt, V., Billaut, J.C., (2002), *Multicriteria scheduling: theory, models and algorithms*, Springer, Berlin.

[31] Tang, C.S., Denardo, E.V. (1988), *Models arising from a flexible manufacturing machine. Part I: Minimization of the number of tool switches*, Operations Research, 36 (5), 767–777.

[32] Tsujimura, Y., Gen, M., (1999), *Parts loading scheduling in a flexible forging machine using an advanced genetic algorithm*, Journal of Intelligent Manufacturing, 12 (3), 413–420.

[33] Voß, S., Woodruff, D.L., (2003), *Introduction to computational optimization models for production planning in a supply chain*, Springer–Verlag, Berlin.

[34] Wolsey, L.A., (1997), *MIP modelling of changeovers in production planning and scheduling problems*, European Journal of Operational Research, 99, 154–165.