Grouping products for the optimization of production processes: A case in the steel manufacturing industry

SILVIA CASADO
Universidad de Burgos
scasado@ubu.es

MANUEL LAGUNA
University of Colorado
laguna@colorado.edu

JOAQUÍN PACHECO
Universidad de Burgos
jpacheco@ubu.es

JULIO C. PUCHE
Universidad de Burgos
jcpuche@ubu.es

Abstract

The optimization of a production process is often based on the efficient utilization of the production facility and equipment. In particular, reducing the time to change from producing one product to another is critical to the fulfillment of demand at a minimum cost. We study the production of steel coils in the context of searching for groups of products with similar characteristics in order to create production batches that minimize the cost of fulfilling production orders originated by a known demand. We formulate the problem as mixed-integer program and develop a heuristic solution procedure. We show that a simplified version of the problem is equivalent to the clique partition problem, which in turn is equivalent to the graph-coloring problem. Computational experiments show that the heuristic procedure is effective in finding high-quality solutions to both the clique partition problem and the original grouping problem that includes additional costs.

Keywords: metaheuristics; combinatorial optimization; manufacturing

February 26, 2020

1. Introduction

Manufacturing processes, in general, consist of three phases: 1) extraction of raw material, 2) conversion into industrial products (primary process), and 3) manufacture of final products (secondary process). Production of steel coils is an example of such a process. First, the raw materials (iron ore, and alloys such as manganese, silicon, carbon, aluminum, and niobium) are extracted. These minerals are combined to produce steel, which through continuous casting is shaped as steel slabs. A strip mill turns the slabs into cold roll coils. Other examples of the three-phase production process include the manufacturing of wooden doors (extraction of wood, wooden board, and door), tiles, aluminum coils, and many others. The characteristics of the final product determine the characteristics of the industrial product. While, in principle, a one-to-one correspondence between the industrial and the final product can be established, there are at least three advantages associated with producing in a setting where the relationship between industrial and final products is one-to-many:

- In industrial processes with large equipment, the cost of stopping and restarting is very high. These costs involve labor for adjusting the equipment to meet different product requirements, energy costs associated with restarting the equipment, production of defective items upon restart, and loss of production time. Therefore, reducing the number of changeovers has a direct cost benefit.
- It has also been documented that the activities associated with changeovers of large production equipment increase the probability of personal injury and machine failure. A reduction of the number of changeovers reduces the risk of machine breakdowns.
- Inventory management can be improved by reducing the number of industrial products. There are fewer items to monitor and the storage system and space can be simplified.

In order to create one-to-many mappings of industrial products to final products, it is necessary to group final products based on a set of relevant attributes. For instance, in the manufacturing of steel coils, the relevant characteristics may be the width, weight, steel grade, and due date of each coil. We illustrate this idea with the following example, in which we focus on the width of each coil. Suppose that we want to produce six types of steel coils with widths of 600, 640, 660, 690, and 710 millimeters (mm). The strip mill reduces the width of a steel slab between 15 and 80 mm. Therefore, a steel slab with a width between 615 and 680 mm is needed to produce a coil with a width of 600 mm. To produce the other four coils, steel slabs with widths of 655-720, 675-740, 705-770, and 725-790 mm are required, respectively. Figure 1 shows the steel slab ranges associated with each coil.

The ranges in Figure 1 can be used to determine pairs of “compatible” steel coils. Two steel coils are compatible if their slab ranges overlap. The overlapping range dictates the width of the steel slab needed to produce the compatible coils. Thus, coils 1 and 2 are compatible because their slab width range overlaps. These two coils could be produced with a steel slab with a width between 655 and 680 mm. Also, coils 1 and 3 overlap and can be produced with a steel slab with a width between 675 and 680 mm. Likewise, coils 2 and 3 overlap and can be produced with a steel slab with a width between 675 and 720 mm. Because coils 1, 2, and 3 are all pairwise compatible, they can all be produced with a single steel slab with a width between 675 and 680 mm, which is the overlapping range of all pairs. Neither coil 4 nor coil 5 can be added to the group because they are not pairwise compatible with coil 1. However, they are

compatible between them and therefore they can form their own group. A steel slab with a width between 725 and 770 mm could be used to produce coils 4 and 5.

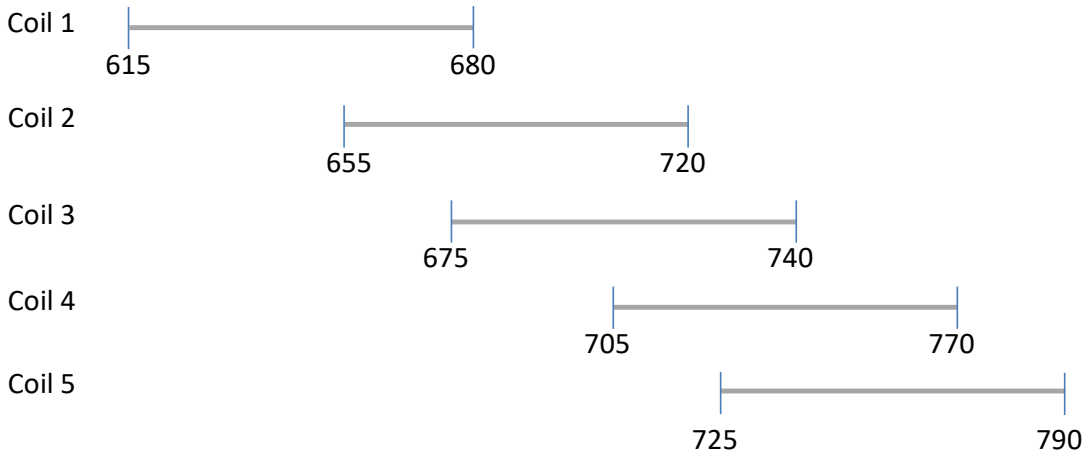

Figure 1. Steel slab ranges for five coil types.

Instead of using five different steel slabs, only two are needed to produce all five coils. Using two instead of five steel slabs results in a cost reduction and the other benefits mentioned above. Although in this example we focused only on the width, in a real setting, pairwise compatibility must be determined based on all characteristics. While there are benefits associated with grouping coils, the tradeoff between these benefits and additional costs must be analyzed. The additional costs are related to an increase of scrap and use of higher steel grades. In our example, if a one-to-one mapping of steel slabs and coils is used, then the waste is a strip of 15 mm regardless of the coil being produced. This is because steel slabs of 615, 655, 675, 705, and 725 mm would be used for coils 1-5, respectively. On the other hand, if the grouping proposed above is used, then the scrap is 75, 35, and 15 mm for coils 1, 2, and 3, respectively. The scrap for coils 4 and 5 would be 35 and 15, respectively.

In terms of steel grade, in a one-to-one setting, the slab matches the grade that the coil requires. In a one-to-many setting, however, the steel slab must be of a grade that matches the maximum grade in the group of coils. This means that some of the coils would have a steel grade that is higher than what is required. To illustrate this, we introduce a graphical representation of pairwise compatibility. In particular, coils are represented by nodes and pairwise compatibility is represented by an edge. Figure 2 shows a set of five coils and their compatibility is represented by the edges that connects them.

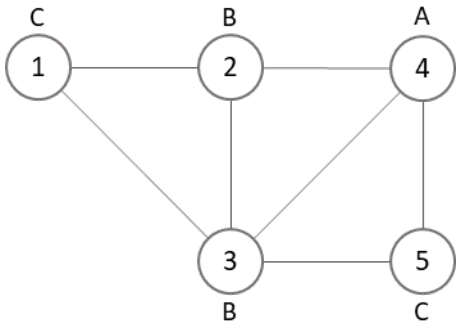

Figure 2. Compatibility graph with steel grades.

The letter next to the nodes in Figure 2 represent the steel grades. Assume that costs are 3, 2, and 1 for grades A, B, and C, respectively. Also, assume that grade A is higher than B and that grade B is higher than C. Figure 3 shows two possible groupings of the coils in Figure 2. The steel grades shown in Figure 3 correspond to the highest grade in the group.

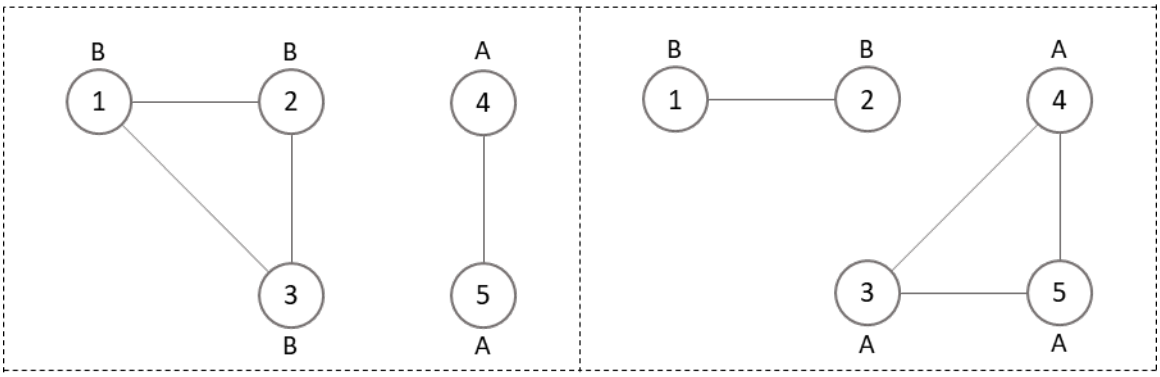

Figure 3. Two possible groupings of coils.

The grouping in the left side of Figure 3 employs three steel slabs of grade B and two slabs of grade A. The total cost is therefore 2*3 + 3*2 = 12. The grouping in the right side of Figure 3 employs three steel slabs of grade A and two slabs of grade B, with a total cost of 3*3 + 2*2 = 13. Clearly, all other things being equal, the grouping in the left side of Figure 3 would be preferred.

We are interested in tackling the problem of producing steel coils at a minimum cost. We consider both the cost of changeover, which is minimized by finding the smallest number of pairwise compatible groups, and the cost of steel grade and scrap, which is minimized by employing slabs that are as close as possible to the minimum requirements. We point out that given a compatibility graph, the problem of minimizing changeovers (ignoring the costs associated with the steel grade and the scrap) is known in the literature as the clique-partitioning problem (Bhasker & Samad, 1991). What we have been calling a compatible group is known as a clique in graph theory. Formally, a clique of a graph $G = (V, E)$, where V is the set of vertices in the graph and E is the set of edges, is a subset W of V such that for every pair of vertices in W there is an edge in E. The clique-partitioning problem consists of finding the smallest number of cliques in a graph such that every vertex in the graph belongs to only one clique. The clique-partitioning of a graph G is equivalent to the graph-coloring of the complement graph G', where each vertex (edge) in G corresponds to an edge (vertex) in G'. The coloring of a graph refers to the problem of finding the smallest number of colors in such a way that each vertex in the graph is assigned a color and that no pair of connected vertices has the same color. To the best of our knowledge, the clique-partitioning problem with a clique cost determined by the product of the maximum node-cost and the size of the clique has not been addressed in the literature.

Our approach simplifies the cost structure of the actual problem that arises in a hot rolling process. In particular, we are not considering a production yield cost associated with an increase in the number of coils to use to fulfill orders. This production cost per coil is independent of weight and width. The cost increases with the number of coils. Therefore, for an order of a given weight, it is better to make coils that are as heavy as possible (within the allowed range) and thus fulfill the order with the smallest number of coils. For instance, suppose that there are two orders, each for a total weight of 1,320 t. Order A is for coils in the range of 10 to 12 t. Order B is for coils in the range of 9 to 11 t. If the orders are fulfilled

separately, then to minimize the number of coils, order A would be fulfilled with 110 coils of 12 t and order B would be fulfilled with 120 coils of 11 t. If instead, to avoid a changeover, we group the orders and fulfill both with coils of 11 t, then the total number of coils would increase by 10, causing an increase in the production cost.

The reason for this simplification is that the managers of the company that engaged us to work on this problem originally wanted groupings of orders that would not cause an "excessive" increase in the production cost. Therefore, they wanted the production cost to be modeled as a constraint. However, since they could not agree in the cost-increase limit, we proposed a bi-objective optimization model in which we would generate a Pareto frontier for the objective of minimizing both the number of cliques and the production cost. This was done with a multiobjective heuristic optimization code that we adapted for the problem. Figure 4 shows a Pareto front that we generated for one of their set of orders.

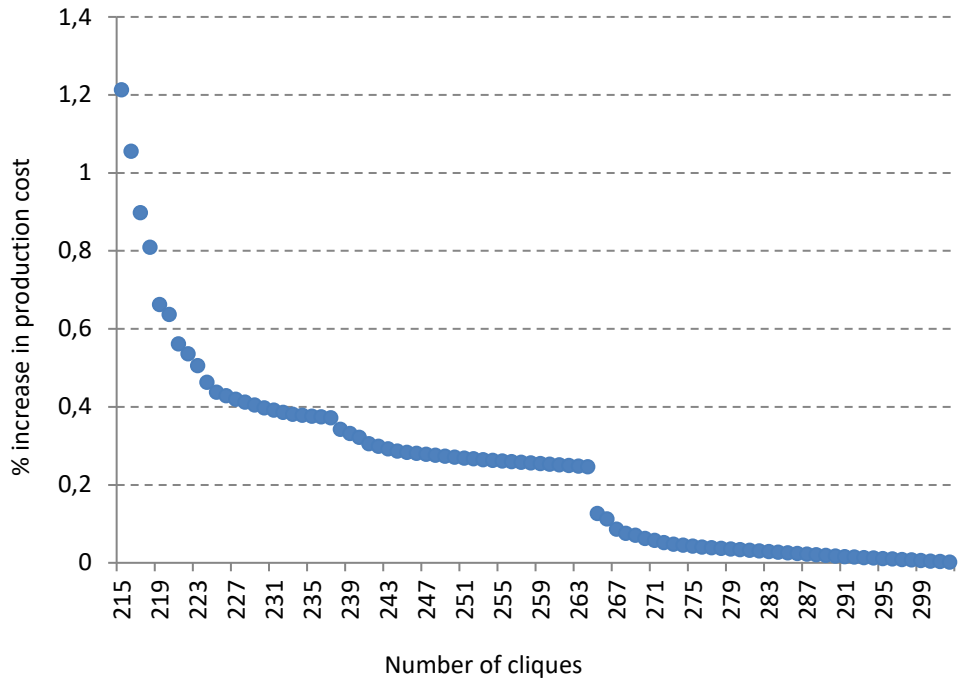

Figure 4. Pareto front approximation for a set of orders.

Figure 4 allowed us to gain two insights on the way the mangers were thinking about this problem. First, they considered an increase of 1.2% in production cost as a small increase when compared to the savings in changeover costs. Second, they found the multiobjective optimization approach and the Pareto front interesting but it became clear that they were focusing on solutions around the minimum number of cliques. In the case of Figure 4, managers were only looking at solutions with 215, 216, or 217 cliques, all of which result in about 1% increase in production cost. This is why we discarded the idea of searching for Pareto fronts and instead we modeled the problem as described in Section 2.

1.1 Relevant Literature

In our literature review we include relevant publications in the management of production as wells as in clique partitioning and graph coloring. Efficient production planning is of great interest to modern manufacturing facilities due to global competition and demand uncertainty (Esmaeilian, Behdad, & Wang, 2016). Cost reduction is critical to improving profit margins and making firms competitive in the global market. The first systematic approaches to production scheduling date back to the mid-1950s. Most of those efforts assumed that the changeover time could be neglected or simply added to the production time. This assumption limited the application of those methods, as the variety of products produced in a single facility increased and the importance of minimizing the time lost to changeovers became clear.

The interest in separating the changeover time from the production time and examining the relationship of lost production time and production assignment and sequences began in the mid-1960s (Wilbrecht & Prescott, 1969; Panwalkar, Dudek, & Smith, 1973). In our work, we are more interested in the idea of grouping products for production than sequencing them. The literature includes some interesting articles related to grouping (jobs, orders, clients, etc.) in production processes. Chen et al. (2005) present a clustering procedure for an order-batching problem in a distribution center with a parallel-aisle layout. They employ an association rule mining approach for order clustering. They provide performance comparisons for various problems between their approach and previously existing heuristics. Anzanello and Fogliatto (2011) cluster product types with similar processing needs into families, such that the efficiency of production programming and resources allocation is maximized. Li et al. (2012) developed a time-series clustering algorithm to create product families out of thousands of items and in doing so facilitate the production planning in a semiconductor plant. In the context of cellular manufacturing, Uddin and Shanker (2002) employ genetic algorithms to tackle the problem of simultaneously assigning process routes (parts) and machines to cells in order to minimize inter-cell movement, while considering multiple routings for each part. Potočnik et al. (2013) describe an approach to organize production cells by means of clustering products into groups with similar properties. These clustering methods are applied to production data from a Slovenian company. Nananukul (2013) considers a customer-clustering problem where customer demands must be satisfied in each period of a planning horizon with limited production and transportation capacity. A practical batching decision problem that arises in the batch annealing operations of the cold rolling stage of steel production is addressed by Tang et al. (2015). This problem is faced by most large iron and steel companies in the world. The problem is to select steel coils from a set of waiting coils to form batches to be annealed in available batch annealing furnaces and to choose a median coil for each furnace. The objective is to maximize the total reward of the selected coils minus the total coil–coil and coil–furnace mismatching cost. Additional recent applications of batching in steel manufacturing are addressed by Ma et al. (2014), Peng et al. (2016), and Tang et al. (2014).

Song (2014) tackles the cast design problem in the steel making industry. A cast is a set of charges, where a charge is a batch of molten steel (i.e., transformed iron ore). Charges in a cast have similar steel grades and, therefore, to minimize changeover time (i.e., maximize production efficiency), it is desirable to configure casts with the largest number of charges. Song approaches the cast design problem in three stages: charge design, grouping, and sequencing. The charge design consists of two sub-problems, 1) slab design (a variant of the bin-packing problem) and 2) half-charge matching. The final stage consists of grouping and sequencing the charges to create efficient casts. The sequencing is formulated as a vehicle routing problem. A commercial solver is applied to mixed-integer programming formulations of small

problem instances. Heuristic optimization is applied to larger instances. The heuristic includes a pre-grouping step that attempts to simplify the problem by identifying compatible orders (based on steel grade transition rules). Cliques are found in a compatibility graph, where an edge between two orders represents their compatibility to be sequenced in the same cast. We point out that this step is only trying to identify cliques and not to solve the clique-partitioning problem, as we propose. Additional relevant literature in steel production is due to Lopez, Carter, and Gendreau (1998), and Tang et al. (2000; 2001).

When dealing with a large number of products (e.g., in the order of ten thousand) in settings where scheduling decisions must consider sequence-dependent changeover times, employing an explicit changeover matrix becomes impractical. Clustering, or grouping, by taking into consideration key product attributes, can significantly reduce the dimensionality of the changeover matrix. Baykasoğlu and Ozsoydan (2018) use this approach for the dynamic scheduling of parallel heat treatment furnaces.

Mass customization in manufacturing systems is employed to remain competitive in a marketplace with shifting demands for products and emerging competition. Mass customization results in an exploding number of part/product variants (or variants for short). These variants require a number of operations, whose sequence could be used to group similar variants, that is, those with similar operational flows. Variants can also be grouped by volume similarity in order to increase machine utilization. Navaei & ElMaraghy (2016) develop a comprehensive similarity coefficient that takes into consideration operations, flows, and volume. The coefficients are used by a clustering algorithm, with the average-linkage criterion, to create groups of variants in a cellular manufacturing environment.

While there is a rich literature on the maximum clique problem, the clique-partitioning problem (CPP) literature is relatively sparse. It starts with the seminal work reported in (Bhasker & Samad, 1991). This work presents two theoretical results associated with the clique-partitioning problem. First, it formulates a new upper bound for the number of cliques in a problem. Then, it shows that an optimal partition includes a maximal clique. It also shows that the clique-partitioning problem is equivalent to the graph-coloring problem and therefore a comparison of the clique-partitioning algorithms developed in the article is made with two of the best graph coloring algorithms known at the time. Several recent publications tackle the clique-partitioning problem (Oosten, Rutten, & Spieksma, 2001; Jaehn & Pesch, 2013; Sundar & Singh, 2017; Brimberg, Janićijević, Mladenović, & Urošević, 2017). Oosten, Rutten, and Spieksma (2001) define the clique-partitioning problem on a graph with edge weights. The problem is to find a partition of the vertices into nonempty, disjoint sets such that these sets form cliques and that the sum of their edge weight is maximum. They present a procedure, called patching, which is able to construct new facets by making use of already-known facet-defining inequalities. A variant of this procedure is shown to run in polynomial time. Jaehn and Pesch (2013) discuss the problem of clustering the vertices of a complete edge-weighted graph. The objective is to maximize the sum of the edge weights within the cliques. Real-life applications of this problem include groupings in flexible manufacturing systems, in biology, and in flight gate assignments. Numerous heuristics and exact approaches as well as benchmark tests have been presented in the literature for this version of the problem. Most exact methods use branch and bound with branching over edges. Jaehn and Pesch (2013) developed tighter upper bounds for each search tree node than those known in the literature at the time; they improved the constraint propagation techniques for fixing edges in each node, and introduced a new branching scheme. Sundar and Singh (2017) define the clique-partitioning problem as finding the minimum number of subsets such that each subset is a clique. The authors developed two metaheuristic techniques based

on evolutionary computation. Both of the proposed approaches are designed in such a way that the grouping structure of the CPP is exploited while generating new solutions. They pay special attention to the design of a neighboring solution generation method utilizing solution components from multiple solutions. The proposed approaches were tested on 37 publicly available DIMACS graph instances. Brimberg et al. (2017) show that the clique-partitioning problem can be reformulated as a maximally diverse grouping problem (MDGP). They modify a skewed general variable neighborhood search (SGVNS) heuristic that was first developed to solve the MDGP. Similarly as with the MDGP, significant improvements over the state of the art are obtained when SGVNS is tested on large-scale instances. This further confirmed the usefulness of a combined approach of diversification afforded with skewed VNS and intensification afforded with the local search in general VNS.

As we mentioned above, the graph-coloring problem (GCP) is related to the CPP. The GCP has many practical applications in areas such as scheduling (de Werra, 1996; Burke, McCollum, Meisels, Petrovic, & Qu, 2007; Ganguli & Siddhartha, 2017; Leighton, 1979; Sabar, Ayob, Qu, & Kendall, 2012; Zais & Laguna, 2016; Wood, 1969; Zufferey, Amstutz, & Giaccari, 2008), timetabling (Yáñez & Ramírez, 2003), manufacturing (Class, 2002), telecommunications (Demange, Ekim, Ries, & Tanasescu, 2014), register allocations (Chaitin, 2004; de Werra, Eisenbeis, Lelait, & Marmol, 1999), air traffic flow management (Barnier & Brisset, 2004), and flight level allocation (Allignol, Barnier, & Gondran, 2012), to name a few. Exact methods for the GCP have also been proposed (Zhou, Li, Huang, & Xu, 2014). However, most practical applications employ metaheuristic methodology (Moalic & Gondran, 2018; Zhou, Hao, & Duval, 2016; Avanthay, Hertz, & Zufferey, 2003; Lü & Hao, 2010). Comprehensive surveys of the GCP research are due to Galinier et al. (2013), Galinier and Hertz (2006), and Malaguti and Toth (2009).

1.2 Contributions

Our contributions are both methodological and practical. We introduce a new problem to the operations research literature. The problem is introduced in the context of production of steel coil, for which the authors developed a decision support system that a company in Northern Spain is currently using. Developing effective and efficient solution methods for this optimization problem yields important benefits, such as significant reductions in cost, risk, personal injury, and warehouse space. Implementation of good solutions also results in improved logistics. Specifically, the main contributions of this work are:

1. We engaged with a company to model a problem that is of extreme relevance to them and that we found applicable to production processes in general.
2. We propose a mathematical formulation of this problem and analyze the relationship between the proposed model and problems defined on graphs. In particular, we explore the connection with the CPP and the GCP.
3. We develop a heuristic method based on tabu search within a multi-start framework. The solution method is configured to find near-optimal solutions in short computational runs. The method is designed to find solution to the clique-partitioning problem and to the original production problem (which includes additional costs).
4. We establish the quality of the solutions obtained by the proposed solution method with extensive computational experiments. We compare the performance of our method with the state-of-the-art of heuristic solvers for the CPP and GCP. For the original production problem,

we assess performance by comparing our results with off-the-shelf optimizers, one based on mathematical programming (Cplex) and one based on metaheuristics (LocalSolver).

The remainder of the article is organized in the same order as we have listed the contributions. That is, we start with the mathematical formulation, we establish relationships with the graph problems, we describe our solution method, and we show our computational experiments.

2. Formulation and Connections with Graph Problems

Assume that there are n products (e.g., steel coils) to be produced in a single facility. For each pair of products, it is known whether they are compatible or not. The compatibility can be determined by a set of characteristics associated with the products, as discussed in the introduction. The problem consists of grouping the products in mutually exclusive and collectively exhaustive subsets. The subsets must be cliques and the collection of cliques should minimize a cost function. There are two costs, one associated with the number of cliques, and a second one associated with the membership in each clique. We define the problem in terms of a graph and then provide a mathematical formulation.

Let $G = (V, E)$ be a graph with n vertices, each of them representing a product (i.e., $|V| = n$). The set of edges represent the product compatibility. Therefore $(i, j) \in E$ if product i is compatible with product j. There is also a weight w_i associated with each vertex i. The weight represents the cost of the material to produce product i. The weight W_k of a clique C_k, which represents the cost of producing all the products in it, is related to the maximum weight of all the products in the clique and it is calculated as follows:

$$W_k = n_k \left(\max_{i \in C_k} w_i \right)$$

where n_k is the number of vertices assigned to clique C_k. All the products in a clique are produced without changes to the production facility. This means that the same material is used to produce all the products in the clique. The single material to be used is the one with the highest grade and therefore it is the most expensive. We formulate the problem as a bi-objective mathematical program with the following three sets of variables:

x_{ik}: A binary variable that equals one if product i belongs to clique C_k

y_k: A binary variable that equals one if clique C_k is not empty

m_k: Maximum weight assigned to clique C_k, i.e., $m_k = \max_{i \in C_k} w_i$

n_k: Number of vertices assigned to clique C_k

The mathematical formulation is:

Minimize $f_1 = \sum_k y_k$ (1)

Minimize $f_2 = \sum_k n_k m_k$ (2)

Subject to

$$x_{ik} \leq y_k \qquad \forall i,k \tag{3}$$

$$\sum_k x_{ik} = 1 \qquad \forall i \tag{4}$$

$$\sum_{i=1}^{n} x_{ik} \leq n_k \qquad \forall k \tag{5}$$

$$\sum_{(i,j)\in E} x_{ik}x_{jk} \geq n_k(n_k - 1)/2 \qquad \forall k \tag{6}$$

$$w_i x_{ik} \leq m_k \qquad \forall i,k \tag{7}$$

$$n_k, m_k \geq 0 \qquad \forall k \tag{8}$$

$$x_{ik} = \{0,1\} \text{ and } y_k = \{0,1\} \qquad \forall i,k \tag{9}$$

The general model considers two objective functions. Objective function (1) minimizes the number of cliques, which corresponds to minimizing the number of product groups and therefore the number of changeovers. The second objective function minimizes the cost of material, calculated as the maximum cost in a group of products multiplied by the number of products in the group. These two objectives are in conflict. The minimization of f_1 calls for creating as few groups as possible, causing an increase in the size of the groups. On the other hand, large groups tend to deteriorate the value of f_2 because material that is more expensive is used to produce more products. In this work, we treat these two objectives hierarchically, with f_1 as the primary objective and f_2 as the secondary objective.

Constraints (3) restrict the assignment of product i to group k only if group k is being used (i.e., if $y_k = 1$). Constraints (4) specify that every product must be assigned to exactly one group. Constraints (5) calculate the number of products assigned to group k. Constraints (6) ensure that each group k is a clique. The number of edges in the group should be equal to the number of pairwise combinations of vertices in the group. Constraints (7) calculate the maximum vertex weight in each group. The model ends with non-negativity constraints for the continuous variables (8) and the binary restrictions for the assignment variables (9).

Constraints (6) may be linearized as follows:

$$\sum_{(i,j)\in E} x_{jk} - (x_{ik} - 1)n \geq n_k - 1 \qquad \forall i,k \tag{10}$$

By eliminating (2) and (7) from the formulation, the problem becomes the *clique-partitioning problem* (also known as the *minimum clique partition*), as defined by Bhasker and Samad (1991). A clique partition of a graph $G = (V, E)$ is a partition of V such that all subsets in the partition consists of vertices that are pairwise adjacent. That is, for every pair of vertices i and j in a subset, $(i, j) \in E$. The problem of finding the minimum cardinality of a clique partition, $\bar{\chi}(G)$, is NP-Hard in general graphs and as hard to approximate as the graph-coloring problem (Burke, Mareček, Parkes, & Rudová, 2010). The relationship between the minimum clique partition and the graph coloring problem is such that $\bar{\chi}(G) = \chi(\bar{G})$, where $\chi(\bar{G})$ is the minimum number of colors needed to color the complement of the graph G.

To the best of our knowledge, the grouping problem that we study here has not been addressed in the literature. The problem can be considered a variant of the CPP with additional constraints and with a secondary objective function.

3. Solution Method

Our solution approach embeds tabu search (TS) in a multi-start framework. It is an iterative process consisting of constructing solutions and then searching for improved solutions following TS strategies. We first describe the solution construction method. A solution is a set of cliques $S = \{C_1, \dots, C_K\}$, where K is the number of cliques. We define U as the set of vertices that have not been selected and therefore are not yet part of the solution. Initially, $U = V$. The construction procedure, creates cliques one at a time. Suppose that clique C_k is being created. We define U' as those vertices in U that are adjacent to clique C_k if C_k is not empty, that is, $U' = \{j \in U: (i,j) \in E, i \in C_k\}$. If C_k is empty, then $U' = U$. We also define L_i as the set of vertices in U' that are adjacent to vertex $i \in U'$. That is, $L_i = \{j \in U': (i,j) \in E\}$. Only the vertices U' are eligible to be added to C_k. If U' is empty but U is not empty, then a new clique must be created. If both U' and U are empty, then the procedure terminates. A pseudocode of this procedure is shown in Algorithm 1.

Algorithm 1. Solution construction.

1. $S = \emptyset, k = 0, U = V$
   do
2. $k = k + 1, C_k = \emptyset, U' = U$
   do
3. $L_i = \{j \in U': (i,j) \in E\} \forall i \in U'$
4. $i^* = argmax\{|L_i|: i \in U'\}$
5. $C_k = C_k + \{i^*\}$
6. $U = U \backslash \{i^*\}$
7. $U' = L_{i^*}$
   until $U' = \emptyset$
   until $U = \emptyset$

The construction procedure outlined in Algorithm 1 is deterministic; that is, for the same input it always produces the same set of cliques. However, for a multi-start procedure, it is desirable to generate diverse solutions that may be subjected to an improvement phase. To this end, we add a greedy function $g(i)$ that, at each step of the construction process, measures the attractiveness of assigning vertex $i \in U'$ to clique C_k. As in Algorithm 1, the attractiveness of vertex i is simply given by its degree, and therefore the greedy function is defined as:

$$g(i) = |L_i|$$

To induce diversification, the selection is made randomly among a candidate list of vertices:

$$LC = \{i \in U' : g(i) \geq \alpha g_{max} + (1 - \alpha) g_{min}\}$$

where g_{max} and g_{min} are the maximum and the minimum values of $g(i)$ for all $i \in U'$. The adjustable parameter α regulates the semi-greedy nature of the procedure. When $\alpha = 1$, the selection process becomes totally deterministic (as in step 4 of Algorithm 1), and, when $\alpha = 0$, the process becomes totally random.

We also experimented with a more complex greedy function that included two terms, one with a normalized value of the degree of the candidate vertex, and another one with a normalized value of a

frequency memory function. Each candidate vertex had a frequency count with respect to all the vertices already assigned to the clique to which it was being considered. A large frequency value indicated that the candidate vertex and the already assigned vertices had been in the same clique frequently. That is, the frequency memory was a simple table in which the element in row i and column j indicated the number of times that vertex i and vertex j were assigned to the same clique during the search. The function then discouraged the assignment of vertices to the same clique as their frequency count increased. An additional parameter was introduced to balance the importance given to this penalty term. Experiments showed that the best results were obtained when the frequency memory information was ignored. Therefore, we simplified the greedy function to focus on the degree of a vertex and control diversification with the value of α.

The solutions generated with the semi-greedy process are subjected to an improvement process based on a short-term memory tabu search (Glover, 1989; Glover, 1990; Glover & Laguna, 1993). This search explores a neighborhood consisting of all the feasible solutions that can be reached by moving one vertex from one clique to another. Only feasible moves are considered, that is the solution after a move should be a set of cliques. A move that takes the search from solution S to solution S' is evaluated using three criteria:

$$\Delta f_1 = f_1(S) - f_1(S')$$
$$\Delta sqr = sqr(S) - sqr(S')$$
$$\Delta f_2 = f_2(S) - f_2(S')$$

where $sqr(S) = \sum_{C_k \in S} n_k^2$. That is, $sqr(S)$ is the sum of the squares of the cardinality of each clique in the solution. The non-tabu move with the maximum value of Δf_1 is selected. If all moves have the same Δf_1 value, then the move with the maximum value of Δsqr is selected. Note that $sqr(S)$ favors solutions with a large variance in the cardinality of each clique. This criterion moves the search toward solutions where some of the cliques are significantly larger than other cliques. This produces a search direction where subsequent moves could eventually reduce the number of cliques (by emptying cliques with few vertices). Consider the following three solutions:

$$S_1 = \{\{1,2,3,4,5,6\}\{7,8,9,10\}\}$$
$$S_2 = \{\{1,2,3,4,5,6\}\{7,8\}\{9,10\}\}$$
$$S_3 = \{\{1,2,3,4\}\{5,6,7\}\{8,9,10\}\}$$

The best solution is S_1, because $f_1(S_1) = 2$ and $f_1(S_2) = f_1(S_3) = 3$. According to the sqr metric, S_2 is better than S_3, since $sqr(S_2) = 6^2 + 2^2 + 2^2 = 44$ and $sqr(S_3) = 4^2 + 3^2 + 3^2 = 34$.

If a move of vertex i from its current clique k to another clique is executed in iteration $iter$, a short term memory in the form of a two-dimensional array ($TabuIter$) is updated with the iteration number in which the vertex will be released from its tabu status:

$$TabuIter(i,k) = iter + tenure$$

The value of $tenure$ is a search parameter that indicates the number of iterations that vertex i is prevented from being moved back to clique k. Therefore, in a future iteration $iter$, the move of vertex i to clique k is classified tabu if $iter \leq TabuIter(i,k)$. This tabu classification is overridden if the move of vertex i to clique k improves upon the incumbent solution. This is the typical tabu search aspiration

criterion. Therefore, the neighborhood, $N(S)$, of a solution S consists of all the solutions that can be reached with non-tabu moves plus any solution that meets the aspiration criterion. The short-term memory tabu search improvement procedure is summarized in Algorithm 2.

In step 1 of Algorithm 2, the solution S generated by the construction procedure is augmented with an empty clique $K+1$. This is done to give the search procedure the flexibility of creating a solution with one more clique. Note that the construction procedure focuses on minimizing the number of cliques but solutions with better values of f_2 may not necessarily be the ones with the best values of f_1. Initialization takes place in steps 2 and 3. Step 5 identifies the best solution in the neighborhood of the current solution. This is where the evaluation criteria Δf_1, Δsqr, and Δf_2 are used. The chosen move is executed in step 6 and the search moves to solution S'. The best solution is updated in step 7. The best solution S^* is with respect to f_1 and f_2 only. That is, the sqr criterion does not play a role in this comparison. Step 8 updates the short term memory and the search ends when $tsiter$ iterations are executed without a change in the best solution found.

Algorithm 2. Solution improvement: Short-term memory tabu search.

1. $S = \{C_1, \dots, C_{K+1}\}$
2. $S^* = S, iter = iter^* = 0$
3. $TabuIter(i,k) = 0 \ \forall (i,k)$
   do
4. $\quad iter = iter + 1$
5. $\quad S' =$ best solution in $N(S)$, reached with move (i,k)
6. $\quad$ Move the search to S'; that is, $S = S'$
7. $\quad$ **if** (S is better than S^*) **then** $S^* = S$ and $iter^* = iter$
8. $\quad TabuIter(i,k) = iter + tenure$

   until $iter - iter^* \geq tsiter$

In our multi-start procedure, the short-term memory ($TabuIter$) is cleared every time the Algorithm 2 is invoked. The multi-start procedure is summarized in Algorithm 3.

Algorithm 3. Multistart procedure.

1. $iter = iter^* = 0$
   do
2. $\quad iter = iter + 1$
3. $\quad S \leftarrow construction(\alpha)$
4. $\quad S' \leftarrow improvement(tenure, tsiter, S)$
5. $\quad$ **if** (S' is better than S^{best}) **then** $S^{best} = S'$ and $iter^* = iter$

   until $iter - iter^* \geq msiter$

The construction in step 3 of Algorithm 3 is the semi-greedy version of Algorithm 1. The improvement in step 4 is the short-term memory tabu search in Algorithm 2. The update of the (overall) best solution in step 5 is made with reference to f_1 and f_2. A solution S^{best} is better than a solution S if $f_1(S^{best}) < f_1(S)$ or $f_1(S^{best}) = f_1(S)$ and $f_2(S^{best}) < f_2(S)$. This is the same criteria that are used to update the best solution during the tabu search (see step 7 in Algorithm 2). The multistart procedure is executed until $msiter$ iterations are performed without a change in the best solution found. We point out that the best solution in Algorithm 3 is the best overall solution while the best solution in Algorithm 2 refers to the best

found within the current iteration of the multistart procedure. That is S^* in Algorithm 2 is a "local best" while S^{best} in Algorithm 3 is the global best. Clearly, those two solutions are the same when Algorithm 2 identifies the overall best solution.

4. Computational Experiments

We perform three main experiments. The first one deals with fine tuning three of the two search parameters, namely, α and $tenure$. The search parameters $tsiter$ and $msiter$ are chosen to produce competitive execution times. The second experiment is a "reality check" that compares the performance of our proposed multistart procedure (MSTS) with a state-of-the-art metaheuristics for the clique partitioning and the graph coloring problems. As we mentioned in Section 2, the literature has favored the development of solutions methods of the graph-coloring problem over the development of procedures to minimize the number of cliques. The goal of our experiment is to show that our procedure is competitive when compared to both graph-coloring and clique-partitioning specialized codes. We then compare the performance of MSTS against commercial general-purpose solvers: Cplex and LocalSolver. The goal of this experiment is to justify the development of MSTS to tackle the bi-objective problem. A customized solution to this problem can only make sense if existing tools are not able to produce similar results. We end by testing our procedure on a real problem instance provided by steel production facility of the company for which we did this work. All the experiments were carried out on a workstation with an Intel processor i7-7700 4.20 GHz and 32 GB of RAM. The codes were programmed in haveObject Pascal (RAD Studio 10.2). We used LocalSolver 8.0 and Cplex 12.8 in our experiments.

We generated 270 problem instances, for which we varied the size and the density of the graph. In particular, we generated five instances of each combination of graph density (0.05, 0.10, and 0.20) and number of vertices ranging from 10 to 1000 (with 18 values of n in total). In a 1000×1000 square, we generated n points with coordinates (x, y), representing the vertices in the graph. We calculated the Euclidean distance between each pair of points and determined the maximum distance. Then, the edge (i, j) between vertex i and vertex j is added to the graph if the distance between the two vertices is less than the maximum distance multiplied by the density. Therefore, the number of edges in the graph increases with the density value[1].

We chose a "training set" of 18 out of the 270 instances for the fine-tuning experiment. This training set consists of one instance and one density of each size, as shown in Table 1.

Table 1. Instances in the training set.

No.	n	Density	Instance	No.	n	Density	Instance	No.	n	Density	Instance
1	10	0.05	1	**7**	50	0.05	2	**13**	150	0.05	4
2	15	0.10	2	**8**	60	0.10	3	**14**	200	0.10	5
3	20	0.20	3	**9**	70	0.20	4	**15**	300	0.20	1
4	25	0.05	4	**10**	80	0.05	5	**16**	400	0.05	3
5	30	0.10	5	**11**	100	0.10	1	**17**	500	0.10	4
6	40	0.20	1	**12**	120	0.20	3	**18**	1000	0.20	5

[1] All of these instances are available here: https://www.ubu.es/metaheuristicos-grinubumet/ejemplos-y-datos-de-problemas

We employ a sequential fine-tuning process, starting with the value of $tenure$ for a fixed value of $tsiter = 10n$. The tabu tenure that we considered were $tenure = n/2, n, 2n, 5n$. One solution for each instance in the training set was found with a single run of Algorithm 2 starting from the solution constructed by Algorithm 1. We compared the average quality of the eighteen solutions obtained with each tabu tenure and selected the best ($tenure = 2n$).

We then adjusted the value of α, the diversification parameter. Recall that α controls the randomization of the construction procedure, whereby a value of α close to one approaches the deterministic construction in Algorithm 1, and a value of α close to zero produces random vertex selections at each step. With fixed values for $msiter = 20$, $tsiter = 10n$, and $tenure = 2n$, we tried $\alpha = 0,0.5,0.9,0.99,1$. The average solution quality, as a measure of the merit of each α value, resulted in $\alpha = 0.99$ as the best value for this parameter.

The large value of α indicates a preference for relatively short candidate lists. These short lists, however, have been shown to be sufficient in terms generating diversity, particularly as the graphs increase in size. This observation coincides with similar experiences reported in the literature (Pacheco, Alfaro, Casado, & García, 2012).

4.1 Performance Comparison against CPP State-of-the-Art

The problem that we are tackling, as formulated in Section 2, can be considered a variant of the CPP with additional constraints and with a secondary objective function. Because there are no specialized methods in the literature for this problem, we first compare the performance of our multi-start tabu search implementation (MSTS) with the state-of-the-art for the CPP. Our literature review revealed that algorithmic development has focused on solving graph-coloring problems as well as procedures that address the CPP directly. Therefore, for comparison purposes, we chose both the best graph-coloring procedure and applied it to $\bar{G}$ and the most recent procedure for the CPP. The chosen graph-coloring procedure is a memetic algorithm, denoted by MEM, due to Moalic and Gondran (2018). Table 2 summarizes the results of the comparison, using the average value of both objective functions as the measure of merit.

The best solutions are shown in bold. When comparing the average objective function values, f^A and f^B, of procedures A and B, the average solution quality of procedure A is better than B if $f_1^A < f_1^B$ or $f_1^A = f_1^B$ and $f_2^A < f_2^B$. The results in Table 2 indicate that the two procedures find the same solutions to problem instances with up to 20 vertices and densities of 0.05 and 0.10. For the instances with 30 to 150 vertices, MSTS dominates MEM on both objective functions. The memetic algorithm is able to find solutions that on average have better f_1 values than the solutions found by the MSTS procedure in instances with more than 300 vertices and a density of 0.2. This may be due to the lower density graph-coloring problem that results from a higher density CPP.

Since MEM is not designed to address f_2, in all the cases when$f_1^{MEM} < f_1^{MSTS}$, we observe that $f_2^{MEM} > f_2^{MSTS}$. Another way of summarizing the results of this experiment is by counting the number of times that each procedure obtains the better solution. Out of the 270 instances, MSTS finds a better solution than MEM 184 times. MEM is able to find better solutions than MSTS in 16 instances. In the remaining 70 instances, both procedures find the same solutions.

We performed two paired tests on instances for which MEM and MSTS obtain different results. The first test considers both objective functions and the second one considers f_1 only. We test the following hypothesis:

$$H_0: \pi \leq 0.5$$
$$H_1: \pi > 0.5$$

Where π is the unknown proportion of times that MSTS obtains better solutions than MEM. In both tests, we reject the null hypothesis. The p-value in the first test is less than 0.0001 and the p-value is 0.0048 for the second test.

The solution times reported in Table 2 correspond to those of MSTS. The execution times for MEM varied from 4 to 14,264 seconds. Therefore, MEM employs at least two orders of magnitude more time than MSTS.

Table 2. Average solution quality comparison between MSTS and MEM.

n	Density	MSTS		Time	MEM		n	Density	MSTS		Time	MEM	
		f_1	f_2		f_1	f_2			f_1	f_2		f_1	f_2
10	0.05	**9.6**	**11.2**	0.01	**9.6**	**11.2**	80	0.05	**54.0**	**104.2**	0.37	54.0	105.6
	0.10	**8.8**	**11.2**	0.01	**8.8**	**11.2**		0.10	**32.2**	**115.0**	0.54	32.2	125.4
	0.20	**6.8**	**13.2**	0.01	6.8	13.4		0.20	**14.4**	**142.6**	0.46	14.4	152.4
15	0.05	**14.4**	**16.8**	0.01	**14.4**	**16.8**	100	0.05	**65.0**	**128.6**	0.71	65.0	132.2
	0.10	**12.0**	**18.0**	0.01	**12.0**	**18.0**		0.10	**36.0**	**154.8**	0.90	36.2	167.8
	0.20	**8.2**	**19.8**	0.01	8.2	21.6		0.20	**15.2**	**178.0**	0.56	15.2	194.0
20	0.05	**18.2**	**20.4**	0.02	**18.2**	**20.4**	120	0.05	**71.6**	**157.8**	1.22	71.6	166.6
	0.10	**16.2**	**21.0**	0.02	**16.2**	**21.0**		0.10	**37.0**	**195.2**	1.84	37.0	207.8
	0.20	**9.8**	**26.4**	0.02	9.8	29.2		0.20	**15.4**	**218.8**	1.19	15.4	234.0
25	0.05	**22.6**	**24.2**	0.03	**22.6**	**24.2**	150	0.05	**83.0**	**208.8**	2.71	83.0	219.6
	0.10	**18.2**	**28.8**	0.02	18.2	29.2		0.10	**40.8**	**249.2**	3.03	41.0	265.6
	0.20	**10.6**	**34.2**	0.03	10.6	36.4		0.20	**16.0**	**276.6**	1.67	16.0	299.0
30	0.05	**27.0**	**34.2**	0.04	27.0	34.4	200	0.05	97.2	282.2	4.98	**97.0**	**303.2**
	0.10	**21.0**	**38.2**	0.04	21.0	40.6		0.10	**42.2**	**346.0**	5.27	42.8	368.0
	0.20	**11.8**	**47.0**	0.06	11.8	51.0		0.20	**16.8**	**369.8**	4.59	16.8	395.6
40	0.05	**33.4**	**49.4**	0.08	33.4	49.8	300	0.05	**114**	**451.2**	13.67	114.8	485.8
	0.10	**24.2**	**54.6**	0.06	24.2	55.4		0.10	**48.6**	**534.6**	15.87	49.4	569.6
	0.20	**12.4**	**63.4**	0.07	12.4	68.0		0.20	17.6	569.8	6.45	**17.4**	**599.0**
50	0.05	**39.8**	**57.0**	0.13	39.8	58.0	400	0.05	**128.4**	**635.6**	27.82	129.4	676.8
	0.10	**25.8**	**70.6**	0.12	25.8	73.6		0.10	**52.8**	**750.6**	28.18	54.0	777.4
	0.20	**14.0**	**79.4**	0.12	14.0	89.8		0.20	18.0	780.2	13.11	**17.6**	**800.0**
60	0.05	**45.6**	**70.2**	0.19	45.6	70.8	500	0.05	**139.6**	**804.6**	51.92	144	864.4
	0.10	**29.6**	**86.2**	0.25	29.6	92.8		0.10	**54.0**	**958.4**	44.29	55.4	984.0
	0.20	**13.2**	**102.0**	0.22	13.2	109.8		0.20	18.8	994.6	24.84	**18.4**	**998.2**
70	0.05	**51.2**	**84.4**	0.27	51.2	86.4	1000	0.05	**172.4**	**1828.4**	900.45	190.4	1871.0
	0.10	**31.0**	**102.4**	0.32	31.0	110.0		0.10	63.4	1957.6	199.03	**62.8**	**1997.4**
	0.20	**14.0**	**116.4**	0.30	14.0	132.6		0.20	21.2	1982.2	105.34	**20.2**	**2000.0**

As discussed in Section 1.1, Sundar and Singh (2017) developed two procedures for the CPP, one based on genetic algorithms (SSGGA) and one based on an artificial bee colony (GABC). The results reported in their article for 37 DIMACS instances are based on 10 runs. MSTS, on the other hand, was executed one time with the stopping criteria specified above and a maximum running time of 3600 seconds. Table 3 shows, for each instance, the number of vertices and the density. For SSGGA and GABC, the table shows the objective function for the best solution in the ten runs (Best), average objective function value (Avg.),

and the average computing time (ACT). The total computational effort of running these procedures is therefore 10 times ACT. For MSTS we show the objective function value of the best solution found (Value) and the solution time in seconds (Time). The values in bold indicate instances in which the corresponding procedure found the best-known solution.

In Table 3, we can observe that only 8 times SSGGA finds a better solution than MSTS and only 4 times GABC finds a better solution than MSTS.For the four instances that GABC produces better solutions than MSTS, the SSGGA solutions are better than those produced by GABC. Due to this overlap, we can state that in only eight instances MSTS fails to produce solutions of equal or better quality as those found by the competing methods. Clearly, MSTS requires more time than the competing procedures; however, the additional computational effort is well employed in the sense that it is able to produce improved outcomes. We note that MSTS was designed and tuned for the problems that we encountered in the real setting that we studied. In those problems, the graphs are rather sparse. We did not observe graphs with a density of more than 0.15, and often the density that we encountered was no more than 0.03. In contrast, only three instances in the DIMACS set have densities of around 0.25. The rest of the instances have densities above 0.48. Nonetheless, MSTS produced extremely competitive results.

Table 3. Comparison between CPP procedure by Sundar and Singh (2017) and MSTS.

Instance			SSGGA			GABC			MSTS	
Name	n	Density	Best	Avg.	ACT	Best	Avg.	ACT	Value	Time
C125.9	125	0.8985	**6**	6.1	1.25	**6**	6	0.08	**6**	0.06
C250.9	250	0.8991	10	10	2.93	10	10	0.5	**9**	1.71
C500.9	500	0.9005	16	16.5	13.44	**15**	15.7	5.44	**15**	1.61
C1000.9	1000	0.9011	26	27.1	63.52	25	25	59.95	**24**	110.03
C2000.9	2000	0.9002	45	45.9	345.56	42	42	527.8	**41**	1890.31
C2000.5	2000	0.5002	**173**	174.1	710.09	**178**	178.9	1167.49	182	1794.74
C4000.5	4000	0.5002	**315**	315.8	5227.38	**321**	322.7	8627.66	329	3198.52
MANN_a27	378	0.9901	**4**	4	2.12	**4**	4	2.59	**4**	0.03
MANN_a45	1035	0.9963	**4**	4	11.82	**4**	4	60.89	**4**	0.51
MANN_a81	3321	0.9988	**4**	4	108.96	**4**	4	2726.52	**4**	21.33
brock200_2	200	0.4963	**25**	25.9	2.47	27	27.4	1.29	27	5.54
brock200_4	200	0.6577	**18**	18.5	2.52	19	19.4	1.04	19	5.13
brock400_2	400	0.7492	**25**	25.5	10.99	**25**	25.4	7.54	**25**	7.45
brock400_4	400	0.7489	**25**	25.9	11.44	**25**	25.4	7.07	**25**	10.31
brock800_2	800	0.6513	**57**	57.7	51.59	58	58.4	72.11	58	1621.50
brock800_4	800	0.6497	**57**	58	46.59	**58**	58.3	74.51	59	118.13
gen200_p0.9_44	200	0.9000	9	9	1.41	**8**	8.6	0.29	**8**	0.14
gen200_p0.9_55	200	0.9000	**7**	7.6	1.63	**7**	7.5	0.23	**7**	1.05
gen400_p0.9_55	400	0.9000	14	14.2	7.03	14	14	3.41	**13**	5.74
gen400_p0.9_65	400	0.9000	13	13.9	6.88	13	13.9	2.74	**12**	2.41
gen400_p0.9_75	400	0.9000	13	13.8	5.83	13	13.1	3.03	**12**	41.07
hamming8-4	256	0.6392	**16**	16	7.03	**16**	16	1.18	**16**	12.97
hamming10-4	1024	0.8289	38	38	62.89	37	37.2	107.48	**35**	83.91
keller4	171	0.6491	**20**	20.2	2.14	21	21.3	0.82	**20**	10.04
keller5	776	0.7515	**44**	45.5	39.98	47	47.1	68.85	**44**	1584.43
keller6	3361	0.8182	100	101.5	1537.98	103	103.1	4459.84	**95**	614.32
p_hat300-1	300	0.2438	66	66.5	3.61	70	70.8	2.56	**65**	266.35
p_hat300-2	300	0.4889	45	45.6	4.07	45	46.4	3.1	**44**	320.75
p_hat300-3	300	0.7445	22	22.5	4.21	22	22.7	3.11	**21**	124.53
p_hat700-1	700	0.2493	135	137.4	19.93	146	147.4	36.63	**133**	467.27
p_hat700-2	700	0.4976	93	94.1	19.29	93	94	35.44	**90**	965.56
p_hat700-3	700	0.7480	44	44.3	25.31	43	44.2	32.86	**41**	654.41
p_hat1500-1	1500	0.2534	271	271.8	120.12	284	285.4	263.04	**257**	1601.79
p_hat1500-2	1500	0.5061	175	176.4	103.6	176	177.7	254.52	**169**	1490.55
p_hat1500-3	1500	0.7536	80	81.2	160.47	79	79.8	290.49	**76**	182.04
DSJC500_5	500	0.5020	**54**	54.4	14.72	**56**	56.7	20.16	57	18.35
DSJC1000_5	1000	0.5002	**96**	97.3	76.08	100	101	143.92	100	3294.70

4.2 Performance Comparison against Cplex

We created a linear, single-objective, model in order to compare the performance of MSTS with a commercial mixed-integer programming solver such as Cplex. The linear model consists of substituting constraint (6) with its linearized version (10). We construct the objective function with a big M method, where M is calculated as follows:

$$M = n\left(\max_{i \in V} w_i\right)$$

The objective function becomes:

Minimize $\quad f_3 = M \sum_k y_k + \sum_k W_k$

The following constraint is added to the model to calculate W_k:

$$W_k \geq w_i n_k - (1 - x_{ik})M \qquad \forall i, k \tag{11}$$

The objective function f_3 gives preference to solutions with minimum number of cliques. The second element of the objective function, i.e., the sum of the weights, acts as a tiebreaker for solutions with the same number of cliques. Since the optimal number of cliques is unknown, we set an upper bound of K cliques that corresponds to the number of cliques found with our heuristic approach.

Table 4 shows the results of the experiment that compares the linearized single-objective model solved with Cplex and MSTS. Both procedures were given a time limit of 3600 seconds. We use the random instances that we generated for the comparison with MEM. However, we only perform experiments with up to 120 vertices, given that Cplex was not able to handle any larger instances. The f_1 and f_2 columns show the average objective function values obtained by each method. The "Sol" column shows the number of instances for which Cplex is able to find at least one feasible solution. The "Opt" column shows the number of times that Cplex is able to find and confirm an optimal solution before the time limit.

Clearly, solving the linearized model with off-the-shelf software is the preferred methods for problems with up to 60 vertices. Even though, Cplex is not able to confirm optimality, MSTS is not able to find better solutions than Cplex for instances with up to 60 vertices. For larger instances, Cplex struggles to find feasible solutions as evidenced by the decreasing trend in the "Sol" values.

Table 4. Comparison between Cplex and MSTS.

n	Density	MSTS			Cplex					MSTS	Cplex
		f_1	f_2	Time	Sol	f_1 (1)	f_2 (1)	Opt	Time (2)	Best	Best
10	0.05	**9.6**	**11.2**	0.01	5	**9.6**	**11.2**	5	0.10	0	0
	0.10	**8.8**	**11.2**	0.01	5	**8.8**	**11.2**	5	0.19	0	0
	0.20	**6.8**	**13.2**	0.01	5	**6.8**	**13.2**	5	0.13	0	0
15	0.05	**14.4**	**16.8**	0.01	5	**14.4**	**16.8**	5	0.56	0	0
	0.10	**12.0**	**18.0**	0.01	5	**12.0**	**18.0**	5	0.63	0	0
	0.20	**8.2**	**19.8**	0.01	5	**8.2**	**19.8**	5	0.39	0	0
20	0.05	**18.2**	**20.4**	0.02	5	**18.2**	**20.4**	5	2.05	0	0
	0.10	**16.2**	**21.0**	0.02	5	**16.2**	**21.0**	5	2.02	0	0
	0.20	**9.8**	**26.4**	0.02	5	**9.8**	**26.4**	5	1.47	0	0
25	0.05	**22.6**	**24.2**	0.03	5	**22.6**	**24.2**	5	5.81	0	0
	0.10	**18.2**	**28.8**	0.02	5	**18.2**	**28.8**	5	8.19	0	0
	0.20	**10.6**	**34.2**	0.03	5	**10.6**	**34.2**	5	4.24	0	0
30	0.05	**27.0**	**34.2**	0.04	5	**27.0**	**34.2**	5	482.55	0	0
	0.10	**21.0**	**38.2**	0.04	5	**21.0**	**38.2**	5	268.56	0	0
	0.20	**11.8**	**47.0**	0.06	5	**11.8**	**47.0**	5	18.72	0	0
40	0.05	**33.4**	**49.4**	0.08	5	**33.4**	**49.4**	4	53.30	0	0
	0.10	**24.2**	**54.6**	0.06	5	**24.2**	**54.6**	3	64.56	0	0
	0.20	**12.4**	**63.4**	0.07	5	**12.4**	**63.4**	4	178.13	0	0
50	0.05	**39.8**	**57.0**	0.13	5	**39.8**	**57.0**	0	-	0	0
	0.10	**25.8**	**70.6**	0.12	5	**25.8**	**70.6**	0	-	0	0
	0.20	**14.0**	**79.4**	0.12	5	**14.0**	**79.4**	4	490.08	0	0
60	0.05	**45.6**	**70.2**	0.19	5	**45.6**	**70.2**	0	-	0	0
	0.10	**29.6**	**86.2**	0.25	5	**29.6**	**86.2**	0	-	0	0
	0.20	**13.2**	**102**	0.22	5	**13.2**	**102.0**	2	1787.14	0	0
70	0.05	**51.2**	**84.4**	0.27	5	**51.2**	**84.4**	0	-	0	0
	0.10	**31.0**	**102.4**	0.32	5	31.0	102.8	0	-	1	0
	0.20	14.0	116.4	0.30	5	**14.0**	**116.0**	1	3302.73	0	1
80	0.05	**54.0**	**104.2**	0.37	5	54.0	105.6	0	-	2	0
	0.10	**32.2**	**115.0**	0.54	5	32.4	117.4	0	-	4	0
	0.20	**14.4**	**142.6**	0.46	4	14.0	141.8	4	1940.23	1	0
100	0.05	**65.0**	**128.6**	0.71	5	77.2	105.4	0	-	4	0
	0.10	**36.0**	**154.8**	0.90	5	38.2	160.6	0	-	5	0
	0.20	**15.2**	**178.0**	0.56	5	15.2	183.2	1	696.13	2	0
120	0.05	**71.6**	**157.8**	1.22	5	96.0	125.8	0	-	5	0
	0.10	**37.0**	**195.2**	1.84	5	49.2	169.2	0	-	5	0
	0.20	**15.4**	**218.8**	1.19	3	15.7	213.7	0	-	5	0
150	0.05	**83.0**	**208.8**	2.71	5	123.2	195.4	0	-	5	0
	0.10	**40.8**	**249.2**	3.03	5	53.2	257.2	0	-	5	0
	0.20	**16.0**	**276.6**	1.67	2	16.5	291.5	0	-	5	0
200	0.05	**97.2**	**282.2**	4.98	1	198.0	85.0	0	-	5	0
	0.10	**42.2**	**346.0**	5.27	1	98.0	50.0	0	-	5	0
	0.20	**16.8**	**369.8**	4.59	0	-	-	0	-	5	0

(1) Average values include only the instances for which Cplex found a feasible solution.

(2) Computational time includes only those instances for which Cplex terminated before the time limit.

4.3 Performance Comparison against LocalSolver

In the interest of assessing the performance of MSTS on both objective functions taken separately, we compare our results with those found with a LocalSolver model. LocalSolver[2] is a well-established optimization platform based on metaheuristic technology. LocalSolver models may include more than one objective function and the solution process treats them hierarchically. The LocalSolver model use for this experiment is shown in the Appendix.

Table 5 shows the comparison of the average quality of the solutions found with MSTS and LocalSolver. Both procedures find the same solutions for problem instances with up to 40 vertices. For low density graphs, LocalSolver is able to match MSTS in problem instances with up to 120 vertices. For larger instances ($n \geq 150$), LocalSolver solutions are such that on average $f_1^{LocalSolver} > f_1^{MSTS}$. However, we note that in all those cases $f_2^{LocalSolver} < f_2^{MSTS}$. This shows the conflicting nature of the two objective functions, in the sense that significantly better values of f_2 are possible at the expense of deteriorating f_1.

Table 5. Average solution quality comparison between MSTS and LocalSolver.

n	Density	MSTS		LocalSolver		n	Density	MSTS		LocalSolver	
		f_1	f_2	f_1	f_2			f_1	f_2	f_1	f_2
10	0.05	**9.6**	**11.2**	**9.6**	**11.2**	80	0.05	**54.0**	**104.2**	**54.0**	**104.2**
	0.10	**8.8**	**11.2**	**8.8**	**11.2**		0.10	**32.2**	**115.0**	33.0	112.2
	0.20	**6.8**	**13.2**	**6.8**	**13.2**		0.20	**14.4**	**142.6**	15.4	133.0
15	0.05	**14.4**	**16.8**	**14.4**	**16.8**	100	0.05	**65.0**	**128.6**	65.2	128.2
	0.10	**12.0**	**18.0**	**12.0**	**18.0**		0.10	**36.0**	**154.8**	37.4	149.4
	0.20	**8.2**	**19.8**	**8.2**	**19.8**		0.20	**15.2**	**178.0**	18.0	152.8
20	0.05	**18.2**	**20.4**	**18.2**	**20.4**	120	0.05	**71.6**	**157.8**	**71.6**	**157.8**
	0.10	**16.2**	**21.0**	**16.2**	**21.0**		0.10	**37.0**	**195.2**	39.8	178.8
	0.20	**9.8**	**26.4**	**9.8**	**26.4**		0.20	**15.4**	**218.8**	19.4	183.8
25	0.05	**22.6**	**24.2**	**22.6**	**24.2**	150	0.05	**83.0**	**208.8**	83.8	205.8
	0.10	**18.2**	**28.8**	**18.2**	**28.8**		0.10	**40.8**	**249.2**	46.8	216.4
	0.20	**10.6**	**34.2**	**10.6**	**34.2**		0.20	**16.0**	**276.6**	22.8	210.6
30	0.05	**27.0**	**34.2**	**27.0**	**34.2**	200	0.05	**97.2**	**282.2**	99.4	274.4
	0.10	**21.0**	**38.2**	**21.0**	**38.2**		0.10	**42.2**	**346.0**	49.4	295.0
	0.20	**11.8**	**47.0**	12.0	46.6		0.20	**16.8**	**369.8**	24.2	310.2
40	0.05	**33.4**	**49.4**	**33.4**	**49.4**	300	0.05	**114.0**	**451.2**	119.2	424.0
	0.10	**24.2**	**54.6**	**24.2**	**54.6**		0.10	**48.6**	**534.6**	62.4	432.4
	0.20	**12.4**	**63.4**	**12.4**	**63.4**		0.20	**17.6**	**569.8**	37.2	423.8
50	0.05	**39.8**	**57.0**	**39.8**	**57.0**	400	0.05	**128.4**	**635.6**	135.4	593.6
	0.10	**25.8**	**70.6**	26.0	69.2		0.10	**52.8**	**750.6**	75.8	549.2
	0.20	**14.0**	**79.4**	14.8	72.8		0.20	**18.0**	**780.2**	45.2	568.8
60	0.05	**45.6**	**70.2**	**45.6**	**70.2**	500	0.05	**139.6**	**804.6**	156.2	697.8
	0.10	**29.6**	**86.2**	**29.6**	**86.2**		0.10	**54.0**	**958.4**	81.8	691.2
	0.20	**13.2**	**102.0**	13.4	100.8		0.20	**18.8**	**994.6**	46.6	738.6
70	0.05	**51.2**	**84.4**	**51.2**	**84.4**	1000	0.05	**172.4**	**1828.4**	220.0	1410.0
	0.10	**31.0**	**102.4**	31.4	100.6		0.10	**63.4**	**1957.6**	127.0	1259.6
	0.20	**14.0**	**116.4**	14.4	113.8		0.20	**21.2**	**1982.2**	-	-

LocalSolver was set up to run a minimum of 5 seconds and no more than twice the time taken by MSTS. For problem instances with one thousand vertices and density of 0.20, LocalSolver was not able to find

[2] LocalSolver is a commercial software developer of optimization and decision support systems (https://www.localsolver.com/)

feasible solutions within the allotted time. With respect to number of best solutions, out of the 270 instances, MSTS found better solutions than LocalSolver in 127 of them, while LocalSolver did not generate a single better solution than MSTS. In 138 instances, both methods found the same solutions, and in five instances, there is no form of comparison because LocalSolver could not find feasible solutions. These results establish the merit of creating a specialized method for the problem that we are trying to solve.

4.4 Experiment with Real Data

In our final experiment, we use a data set from the company consisting of 478 orders. We first established the order compatibility, using the criteria mentioned in the introduction. This resulted in a graph with 3,037 edges. Therefore, this real problem instance has a density of about 0.03. Figure 5 shows the changes of the value of f_1 for the best solution found throughout the search. The profile of this plot is typical of a metaheuristic search. A well-designed search quickly improves upon a starting point and then, as the quality of the best solution improves, the time between improvements increases.

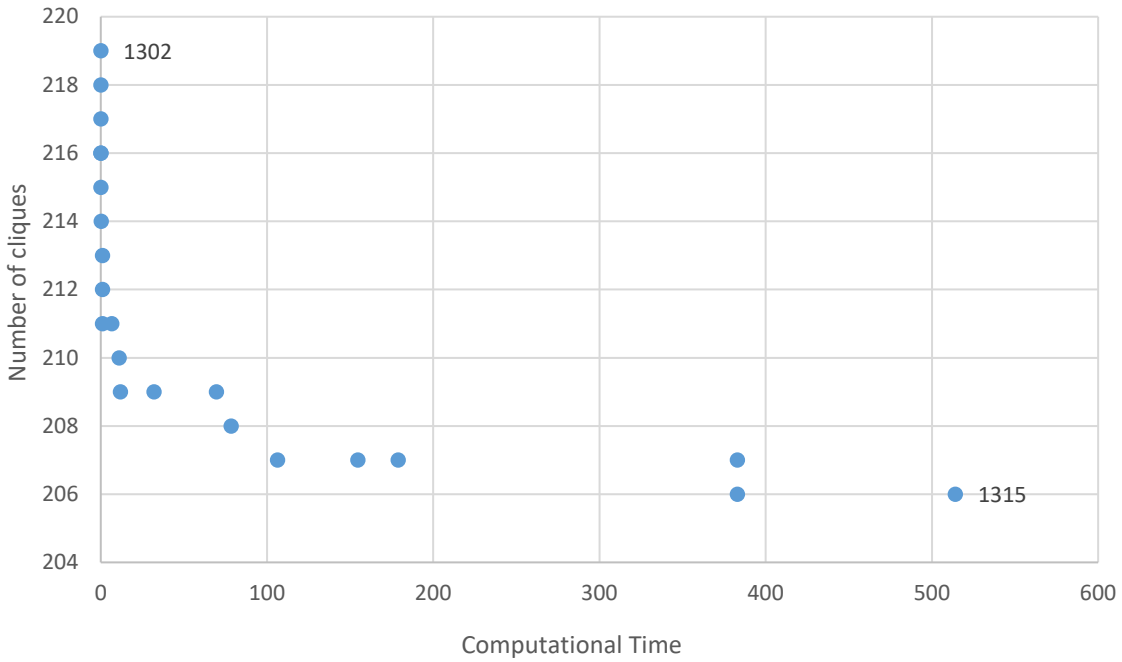

Figure 5. Evolution of the best solution value throughout the search.

The search started from a solution with objective function values of $(f_1, f_2) = (219,1302)$ and finished with the solution with objective function values of $(f_1, f_2) = (206,1315)$. We could not obtain a feasible solution for this problem when solving the linearized model with Cplex. We ran LocalSolver with a time limit of 3600 seconds and obtained a solution with objective function values of $(f_1, f_2) = (208,1242)$. Given the cost of performing changeovers, the reduction from fulfilling each order individually to

combining then into groups represents significant savings for the company. Our procedure is able to reduce the number of changes from 478 (that is, each order in its own group) to 206.

5. Conclusions and Extensions

Many manufacturing processes consist of three production stages. The second stage, known as the primary process, represents a significant fraction of the total manufacturing cost. We described a model that aims at reducing these costs by grouping final products. The main idea is to create groups of final products that can all be manufactured from identical primary products.

The grouping of products yields great benefits. In particular, it enables production continuity by minimizing interruptions. The cost reductions associated with fewer changeovers is significant, as it includes increased production time and a decreased machine repairs and workforce accidents. Product grouping also increases logistic efficiency, by simplifying inventory management and provider operations. Our model seeks to minimize the number of groups (cliques) and it also takes into consideration specific production costs associated with the grouping of products, i.e., the cost of the raw material and waste.

To the best of our knowledge, the optimization problem, as we modeled, with two hierarchical objective functions has not be addressed in the literature. However, we discussed the connections of our optimization problem with the clique partitioning problem and the graph coloring problem that result from dropping the second objective function (i.e., the one related to the weight of each clique). To find high-quality solutions for problems with hundreds of products, we developed a specialized solution procedure based on semi-greedy constructions and short-term memory tabu search. We assessed the performance of the proposed method by comparing our results with those from specialized state-of-the-art clique partitioning and graph coloring methods as well as general-purpose mixed-integer programming and metaheuristic solvers. The experiments showed the merit of our proposal.

Although this was not the original purpose for our work, a byproduct of our efforts is the development of an alternative method for finding high-quality solutions to CPP and graph-coloring instances. Finally, we believe that our solution method is flexible enough to accommodate additional cost considerations as well as different approaches to handle the conflicting objective functions. For instance, the solution procedure could be easily adapted to treat the model as a bi-objective optimization problem.

Acknowledgments

This work was partially supported by FEDER funds and the Spanish Ministry of Economy and Competitiveness (Projects ECO2013-47129-C4-3-R and ECO2016-76567-C4-2-R), the Regional Government of Castilla y León, Spain (Project BU329U14 and BU071G19), and the Regional Government of Castilla y León and FEDER funds (Project BU062U16).

References

Allignol, C., Barnier, N., & Gondran, A. (2012). Optimized flight level allocation at the continental scale. *International Conference on Research in Air Transportation*, (pp. 22-25). Berkeley, California, USA.

Anzanello, M. J., & Fogliatto, F. S. (2011). Selecting the best clustering variables for grouping mass-customized products involving workers' learning. *International Journal of Production Economics, 130*(2), 268-276.

Avanthay, C., Hertz, A., & Zufferey, N. (2003). A variable neighborhood search for graph coloring. *European Journal of Operational Research, 151*(2), 379–388.

Barnier, N., & Brisset, P. (2004, August). Graph coloring for air traffic flow management. *Annals of Operations Research, 130*(1-4), 163-178.

Baykasoğlu, A., & Ozsoydan, F. B. (2018). Dynamic scheduling of parallel heat treatment furnaces: A case study at a manufacturing system. *Journal of Manufacturing Systems, 46*, 152-162.

Bhasker, J., & Samad, T. (1991). The clique-partitioning problem. *Computers and Mathematics with Applications, 22*(6), 1-11.

Brimberg, J., Janićijević, S., Mladenović, N., & Urošević, D. (2017). Solving the clique partitioning problem as a maximally diverse grouping problem. *Optimization Letters, 11*(6), 1123-1135.

Burke, E. K., Mareček, J., Parkes, A. J., & Rudová, H. (2010, September). A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research, 179*(1), 105-130.

Burke, E., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007, January). A graph-based hyperheuristic for educational timetabling problems. *European Journal of Operational Research, 176*(1), 177-192.

Chaitin, G. J. (2004, April). Register allocation and spilling via graph coloring. *ACM SIGPLAN Notices, 39*(4), 66-74.

Chen, M.-C., Huang, C.-L., Chen, K.-Y., & Wu, H.-P. (2005). Aggregation of orders in distribution centers using data mining. *Expert Systems with Applications, 28*, 453-460.

Class, C. (2002). Bag rationalisation for a food manufacture. *Journal of the Operational Research Society, 5*, 544-551.

de Werra, D. (1996). Extensions of coloring models for scheduling purposes. *European Journal of Operations Research, 92*(3), 474-492.

de Werra, D., Eisenbeis, C., Lelait, S., & Marmol, B. (1999, July). On a graph-theoretical model for cyclic register allocation. *Discrete Applied Mathematics, 93*(2-3), 191-203.

Demange, M., Ekim, T., Ries, B., & Tanasescu, C. (2014). On some applications of the selective graph coloring problem. *European Journal of Operational Research, 240*(2), 307-314.

Esmaeilian, B., Behdad, S., & Wang, B. (2016). The evolution and future of manufacturing: A review. *Journal of Manufacturing Systems, 39*, 79-100.

Galinier, P., & Hertz, A. (2006, September). A survey of local search methods for graph coloring. *Computers and Operations Research, 33*(9), 2547-2562 .

Galinier, P., Hamiez, J.-P., Hao, J.-K., & Porumbel, D. (2013). Recent advances in graph vertex coloring. In I. Zelinka, V. Snásel, & A. Abraham (Eds.), *Handbook of Optimization* (pp. 505-528). Berlin: Springer.

Ganguli, R., & Siddhartha, R. (2017). A study on course timetable scheduling using graph coloring approach. *International Journal of Computational and Applied Mathematics, 12*(2), 469-485.

Glover, F. (1989). Tabu Search - Part I. *INFORMS Journal on Computing, 1*(3), 190-206.

Glover, F. (1990). Tabu Search - Part II. *INFORMS Journal on Computing, 2*(1), 4-32.

Glover, F., & Laguna, M. (1993). *Tabu Search.* New York: Springer.

Jaehn, F., & Pesch, E. (2013). New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics, 161*(13-14), 2025-2037.

Leighton, F. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards, 84*(6), 489-506.

Li, B., Li, J., Li, W., & Shirodkar, S. A. (2012). Demand forecasting for production planning decision-making based on the new optimised fuzzy short time-series clustering. *Production Planning & Control, 23*(9), 663-673.

Lopez, L., Carter, M. W., & Gendreau, M. (1998). The hot strip mill production scheduling problem: A tabu search approach. *European Journal of Operational Research, 106*(2-3), 317–335.

Lü, Z., & Hao, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research, 203*(1), 241-250.

Ma, T., Luo, X., & Chai, T. (2014). Modeling and hybrid optimization of batching planning system for steelmaking-continuous casting process. *IEEE/CAA Journal of Automatica Sinica, 1*(2), 113-126.

Malaguti, E., & Toth, P. (2009, December). A survey on vertex coloring problems. *International Transactions in Operations Research, 17*(1), 1-34.

Moalic, L., & Gondran, A. (2018). Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics, 24*(1), 1-24.

Nananukul, N. (2013). Clustering model and algorithm for production inventory and distribution problem. *Applied Mathematical Modelling, 37*(24), 846-9857.

Navaei, J., & ElMaraghy, H. (2016). Grouping part/product variants based on networked operations sequence. *Journal of Manufacturing Systems, 38*, 63-76.

Oosten, M., Rutten, J. H., & Spieksma, F. C. (2001). The clique partitioning problem: Facets and patching facets. *Networks, 38*(4), 209-226.

Pacheco, J., Alfaro, E., Casado, S., & García, N. (2012). A GRASP method for building classification trees. *Expert Systems with Applications, 39*(3), 3241-3248.

Panwalkar, S. S., Dudek, R. A., & Smith, M. L. (1973). Sequencing research and the industrial scheduling problem. In S. E. Elmaghraby (Ed.), *Symposium on the Theory of Scheduling and Its Applications. Lecture Notes in Economics and Mathematical Systems (Operations Research, Computer Science, Social Science)* (Vol. 86, pp. 29-38). Berlin, Heidelberg: Springer.

Peng, K., Zhang, K., You, B., Dong, J., & Wang, Z. (2016). A quality-based nonlinear fault diagnosis framework focusing on industrial multimode batch processes. *IEEE Transactions on Industrial Electronics, 63*(4), 2615-2624.

Potočnik, P., Berlec, T., Starbek, M., & Govekar, E. (2013). Self-organizing neural network-based clustering and organization of production cells. *Neural Computing and Applications, 22*(1), 113-124.

Sabar, N. R., Ayob, M., Qu, R., & Kendall, G. (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence, 37*(1), 1-11.

Song, S. H. (2014). An integrated formulation for hierarchical cast design problems in the steel making industry. *International Journal of Production Research, 52*(5), 1443-1454.

Sundar, S., & Singh, A. (2017). Two grouping-based metaheuristics for clique partitioning problem. *Applied Intelligence, 47*(2), 430-442.

Tang, L., Liu, J., Rong, A., & Yang, Z. (2000). A mathematical programming model for scheduling steelmaking-continuous casting production. *European Journal of Operational Research, 120*(2), 423-435.

Tang, L., Liu, J., Rong, A., & Yang, Z. (2001). A review of planning and scheduling systems and methods for integrated steel production. *European Journal of Operational Research, 133*(1), 1-20.

Tang, L., Meng, Y., Chen, Z. L., & Liu, J. (2015). Coil batching to improve productivity and energy utilization in steel production. *Manufacturing & Service Operations Management, 18*(2), 262-279.

Tang, L., Wang, G., & Chen, Z.-L. (2014). Integrated charge batching and casting width selection at Baosteel. *Operations Research, 62*(4), 772-787.

Uddin, M. K., & Shanker, K. (2002). Grouping of parts and machines in presence of alternative process routes by genetic algorithm. *International Journal of Production Economics, 76*, 219-228.

Wilbrecht, J. K., & Prescott, W. B. (1969). The influence of setup time on job shop performance. *Management Science, 16*(4), B274-B280.

Wood, D. C. (1969, January). A technique for coloring a graph applicable to large-scale timetabling problems. *The Computer Journal, 12*(4), 317-319.

Yáñez, J., & Ramírez, J. (2003). The robust coloring problem. *European Journal of Operational Research, 148*(3), 546-558.

Zais, M., & Laguna, M. (2016). A graph coloring approach to the deployment scheduling and unit assignment problem. *Journal of Scheduling, 19*(1), 73-90.

Zhou, Y., Hao, J.-K., & Duval, B. (2016, December). Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications, 64*, 412-422.

Zhou, Z., Li, C. M., Huang, C., & Xu, R. (2014, November). An exact algorithm with learning for the graph coloring problem. *Computers & Operations Research, 51*, 282-301.

Zufferey, N., Amstutz, P., & Giaccari, P. (2008). Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling, 11*(4), 263-277.

Appendix

The LocalSolver model uses a single set of binary decision variables. These variables assign vertices to cliques. The maximum number of cliques K is an input parameter. In our experiments, we set K as twice the value found by MSTS. This gives enough flexibility to LocalSolver. The assignment constraints are the same as the model in Section 2. The clique constraints are non-linear, which LocalSolver is able to handle. The `n[k]` values are calculated from the decision variables and `c[i][j] = 1` if there is an edge between i and j. The number of cliques in a solution are all those for which `n[k] > 0`. The maximum weight in each clique is calculated with a max function and the decision variables. The two objective functions are declared in hierarchical order. In the interest of briefness, we do not include the input and output functions but the entire LSP file is available from the authors upon request.

```
function model()
{
  // x[i][k] = 1 if item i is assigned to clique k. K is the maximum number of cliques
  x[i in 0..nodes-1][k in 0..K-1] <- bool();

  // Assignment constraints
  for[i in 0..nodes-1]
    constraint sum[k in 0..K-1] (x[i][k]) == 1;

  // Clique constraints
  n[k in 0..K-1] <- sum[i in 0..nodes-1] (x[i][k]);
  for[k in 0..K-1]
    constraint sum[i in 0..nodes-1][j in 0..nodes-1]
    (c[i][j] * x[i][k] * x[j][k]) >= n[k] * (n[k] - 1) / 2;

  // Objective functions are the total number of cliques and the total weight
  numberOfCliques <- sum[k in 0..K-1] (n[k] > 0);
  m[k in 0..K-1] <- max[i in 0..nodes-1](w[i] * x[i][k]);
  totalWeight <- sum[k in 0..K-1](n[k] * m[k]);
  minimize numberOfCliques;
  minimize totalWeight;
}
```