Solving the Wire-Harness Design Problem at a European Car Manufacturer

Marie-Sklaerder Vié[a], Nicolas Zufferey[a,b,*], Jean-François Cordeau[b,c]

[a] *Geneva School of Economics and Management, GSEM - University of Geneva, Switzerland*

[b] *Groupe d'Études et de Recherche en Analyse des Décisions, Montréal, Canada*

[c] *HEC Montréal, Canada*

Abstract

In many industries, increasing competition and high customer expectations compel companies to widen their product offering. Car manufacturers face the same issue and tend to produce an increasingly large variety of models, with more and more electrical options. This leads to the design of a huge number of different electrical wire harnesses, but the use of component commonality can help reduce this number. This work, initiated in collaboration with a major European car manufacturer, provides a new mathematical formulation of the problem and several solution methods. Due to its size, the problem cannot be solved to optimality by exact methods. Hence, we introduce greedy heuristics, a descent method and a variable neighborhood search metaheuristic that provide high quality solutions in reasonable computing time. Results show that the use of these solution methods yields important cost savings with respect to the current practice of the company.

Keywords: Combinatorial Optimization, Metaheuristics, Component Commonality, Car Wire Harness.

1. Introduction

In many industries, fierce competition forces companies to increase their product variety while keeping costs down. Car manufacturers are no exception and tend to increase their product range every year (Statista.com, 2017), which leads to complexity management issues. Each car has numerous different electrical options (e.g., lamps, wipers, radio) and usually contains between one and two kilometers of wire. Due to the wide variety

*Corresponding author: UniMail, Boulevard du Pont-d'Arve 40, 1211 Geneva 4, Switzerland

Email addresses: marie-sklaerder.vie@unige.ch (Marie-Sklaerder Vié), n.zufferey@unige.ch (Nicolas Zufferey), jean-francois.cordeau@hec.ca (Jean-François Cordeau)

of vehicles, car manufacturers have a large number of different electrical kits to build. Indeed, they could make as many wire harnesses (where a wire harness is made of a set of modules, each module having a different presence cost if it appears in the wire harness) as there are electrically different cars, but this brings diversity management costs, called the *diversity tax*. Alternatively, they could make a few standardized wire harnesses, rich enough to suit different vehicles, even if unnecessary modules are set up on many cars, which has also an important cost, called the *give-away*. In any case, some constraints have to be satisfied. On the one hand, some modules require the presence of other modules (*implication* constraints) and, on the other hand, some modules are incompatible (*exclusion* constraints). The goal of this work is to find a balance between these two extreme solutions, while satisfying the demand as well as the implication and exclusion constraints.

This work is motivated by a collaboration with a major European car manufacturer (henceforth referred to as the *Company* because of a non-disclosure agreement) that faces tens of thousands of demands for different wire harnesses to produce. This makes exact methods unsuited. Therefore, heuristics and metaheuristics appear to be the most appropriate for our case. Two greedy heuristics, a descent algorithm and a variable neighborhood search are proposed here. These algorithms are tested on real data provided by the Company. A simpler wire-harness design problem, where the wire harnesses have to be chosen among the demands instead of being designed, was solved exactly for instances with up to a thousand demands in Briant and Naddef (2004). In addition, another wire-harness design problem, with different costs than those considered here, was solved by exact algorithms in Thonemann and Brandeau (2000). However, these methods are limited to much smaller instances (with up to hundreds of demands)

The contribution of this paper is threefold. First, we formulate the *Wire-Harness Design Problem* (WHDP), based on our collaboration with the Company. Second, we introduce generalizable and flexible solution methods that could be applied to related problems arising in different contexts or industries. Third, we report computational results on real data from the Company showing that there is an important potential for savings.

The remainder of the paper is organized as follows. Section 2 formally defines the problem and uses a small example to illustrate some of its features. A literature review is presented in Section 3, where related problems are discussed. An integer linear programming model, and a method used to decompose the problem into smaller subproblems with fewer constraints, are proposed in Section 4, whereas solution methods are described in Section 5. The results are presented in Section 6. Finally, conclusions follow in Section 7.

2. Problem definition

We are given a set $\mathcal{M}$ of all possible modules, where each module corresponds to an electrical option for a vehicle, and a set $\mathcal{D}$ of demands, where each demand $d \in \mathcal{D}$ is a list $\mathcal{M}_d$ of the different modules required and a number Q_d of vehicles to produce. Producing each demand separately would be too expensive for the Company, because of the huge resulting diversity tax. At the other extreme, using a unique wire harness covering all demands would be extremely costly because of the give-away. It would also be impractical because of exclusion constraints.

An *envelope* is a given set of modules that is usually large enough to cover several demands. Each demand belongs to exactly one envelope, i.e., all the modules required in a demand belong to its assigned envelope. The envelopes express technical barriers (e.g., right or left steering), which means that two different envelopes cannot be grouped together as one wire harness. In other words, the envelopes represent implicit exclusion constraints. The set of envelopes is denoted by $\mathcal{E}$, and is given as an input. The problem is to design a set of *references*, where each reference is covered by exactly one envelope. These references must cover all the demands, but will have a smaller production cost than the envelope set (indeed, the give-away will be reduced, but the diversity tax will be augmented). The references have to respect some implication and exclusion constraints for their modules (i.e., if some modules are present in a reference, others must be too, or others cannot be). Note that the give-away can be a non-linear function, because some modules can have different costs if they are together or not in a reference, and some groups of modules have a fixed cost if one or more modules of this group are in a reference. The resulting combinatorial optimization problem (WHDP) consists in building a reference set $\mathcal{S}$ that minimizes the production cost, which is defined as the give-away plus the diversity tax.

As the give-away increases with the cardinality of $\mathcal{S}$, while the diversity tax decreases with it, the production cost is a convex function. Therefore, if this function is known, its minimum is relatively easy to find. If the cardinality of $\mathcal{S}$ is set to a given value N, the associated diversity tax is known, and the remaining problem is to minimize the give-away function. This kind of decomposition (i.e., optimizing each level independently) has proven to be efficient for many optimization problems. For instance, the graph coloring problem is solved much more efficiently by considering k different colors, and then making k increase or decrease depending on the results found (Malaguti and Toth, 2010). In addition, this decomposition allows the Company to have a solution for each possible value N of $|\mathcal{S}|$, which gives them more flexibility in production planning if the demands change.

Using the fact that each demand is covered by only one envelope, and that envelopes represent exclusion

constraints (i.e., the modules of any two demands covered by different envelopes are incompatible), the WHDP can be decomposed into different subproblems, one for each envelope. This can be done by grouping the demands covered by the same envelope. Formally, for each demand $d \in \mathcal{D}$, one and only one envelope $e \in \mathcal{E}$ covers d. It means that a reference r cannot cover two demands that are not covered by the same envelope. For every envelope $e \in \mathcal{E}$, the set of demands covered by e is denoted as $\mathcal{D}_e = \{d \in \mathcal{D} \mid e \text{ covers } d\}$. As a consequence, the full problem can be decomposed into subproblems restricted to $\mathcal{D}_e$ instead of $\mathcal{D}$. These subproblems will only be linked by the constraint on the total number N of references to use for the whole instance. If a solution $\mathcal{S}_e$ is found for each subproblem using a number N_e of references, a solution $\mathcal{S}$ of the WHDP is therefore $\mathcal{S} = \bigcup_e \mathcal{S}_e$, with $\sum_e N_e = N$.

An example with ten modules (meaning that $\mathcal{M} = \{m_1 \dots m_{10}\}$) is now presented. The first important information is of course the list of all the demands. A demand d is a set of modules that the corresponding cars should have. Note that each demand respects the implication and exclusion constraints. Table 1 presents a possible set of demands.

Modules		m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	Q_d
Demands	d_1				√		√	√	√	√		2
	d_2			√	√		√		√	√		1
	d_3	√		√	√		√	√	√		√	4
	d_4		√		√		√				√	5
	d_5	√				√	√	√	√	√		1
	d_6		√			√	√		√	√		2
	d_7					√	√	√		√	√	5
	d_8	√	√			√	√				√	4

Table 1: Demands

The implication and exclusion constraints are presented in Table 2. The symbol $\rightarrow$ denotes implication, and the symbol $\oplus$ is an exclusive "OR", meaning that one and only one of the two variables involved has to be true. The first constraint means that if modules m_1 and m_2 are both in a reference, then the module m_3 has to be present as well. The second constraint means that if m_6 is in a reference, it has to be either with m_4 or with m_5. The third constraint means that either m_8 and m_9 are both in the reference and m_{10} is not, or they are not but m_{10} is.

Constraints	Implication	$m_1 \wedge m_2 \rightarrow m_3$
		$m_6 \rightarrow m_4 \vee m_5$
	Exclusion	$(m_8 \wedge m_9) \oplus m_{10}$

Table 2: Constraints

Table 3 describes the envelopes. For instance, envelope e_1 includes all the modules except m_5 and m_{10}. Note that they can express implicit exclusion constraints. In this example, as m_4 and m_5 are never together in an envelope, they can never be together in a reference (as each reference has to be covered by an envelope). The constraint $m_4 \oplus m_5$ is deduced.

Modules		m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	Demands covered
Envelopes	e_1	√	√	√	√		√	√	√	√		d_1, d_2
	e_2	√	√	√	√		√	√	√		√	d_3, d_4
	e_3	√	√	√		√	√	√	√	√		d_5, d_6
	e_4	√	√	√		√	√	√		√	√	d_7, d_8

Table 3: Envelopes

Table 4 presents the different presence costs of each module or group of modules in a reference. Some modules have different costs if they are grouped with other modules or not. In this example, m_6 has a cost of 5 if it is with m_4 in a reference, but a cost of 2 if it is with m_5. The explicit implication (resp. implicit exclusion) constraints $4 \rightarrow 6$ and $5 \rightarrow 6$ (resp. $4 \oplus 5$) have to be satisfied. In other words, m_6 can only be selected with m_4 or m_5 (but not both), and m_4 and m_5 are incompatible. Moreover, some groups of modules have a common cost if one or more of the involved modules are selected. For instance, there is a cost of 0.01 if either m_7 or m_8 or both are present in a wire harness.

Modules	m_1	m_2	m_3	$m_4 \wedge m_6$	$m_5 \wedge m_6$	$m_7 \vee m_8$	m_9	m_{10}
Costs	4.98	4.98	3.17	5	2	0.01	0.95	0.77

Table 4: Costs of the different modules or groups of modules

A solution with six references is given in Table 5. Each demand is covered by a single reference. More precisely, a demand can be covered by several references, but one has to be chosen (the one that gives the smallest give-away). For instance, the demands d_1 and d_2 are covered by the reference r_1. Each reference is covered by an envelope (e.g., r_2 and r_3 are covered by e_2). Observe that the implication and exclusion constraints are all satisfied.

3. Literature review

The WHDP has common features with several optimization problems, namely the *Generalized Assignment Problem* (Sethanan and Pitakaso, 2016), the *p-Median Problem* (Mladenović et al., 2007), the *Set Covering Problem* (Al-Shihabi et al., 2015) and the *Component Commonality Problem* (Labro, 2004). The literature

Modules		m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	Covered demands	Covering envelope
References	r_1			√	√		√	√	√	√		d_1, d_2	e_1
	r_2	√		√	√		√	√	√		√	d_3	e_2
	r_3		√		√		√				√	d_4	e_2
	r_4	√	√	√		√	√	√	√	√		d_5, d_6	e_3
	r_5					√	√	√		√	√	d_7	e_4
	r_6	√	√	√		√	√				√	d_8	e_4

Table 5: Solution (reference set)

concerning each of these four problems will be briefly reviewed in each of the four following paragraphs. These problems are known to be NP-hard (Garey and Johnson, 1979). Exact solution methods are limited to solving small instances, and they are not appropriate for the problem faced by large car manufacturers. Therefore, heuristics and mostly metaheuristics have been used to solve them. The reader is referred to Gendreau and Potvin (2010) for a survey on metaheuristics and to Zufferey (2012) for guidelines on how to design them efficiently.

First, the WHDP can be seen as a *Generalized Assignment Problem* (GAP), which is also a generalization of the *Multiple Knapsack Problem*. The GAP is about assigning objects to a fixed number of resources. Seeing the demands (resp. the references) as the objects (resp. the resources), the GAP will assign all the demands to a fixed number of references, which is the goal here. In contrast with the GAP, the WHDP has no capacity constraints but a more complex cost function. Due to the complexity of the GAP, exact methods have only been able to solve instances with up to 200 objects and 20 resources (Sethanan and Pitakaso, 2016), which is far from what the Company is handling (tens of thousands of demands). Some greedy algorithms have been developed for the GAP, where weighted functions are designed to measure the desirability of assigning an object to a resource, and then the assignments are performed according to a decreasing order of desirability (Romeijn and Morales, 2000). Lagrangian heuristics were also proposed for the GAP (Jeet and Kutanoglu, 2007; Litvinchev et al., 2010). They consist in performing a Lagrangian relaxation of the problem before using subgradient optimization or a greedy heuristic to repair the relaxed solution. Among the local search methods, one can mention large scale neighborhood search using the shift move and adaptive weights (Yagiura et al., 2004). In the same category, tabu search algorithms have been widely used: with adaptive weights using medium-term memory (Diaz and Fernández, 2001), using a parallel computing environment (James et al., 2009), or employing neighborhoods defined by ejection chains (Laguna et al., 1995). Various evolutionary approaches were also developed for the GAP: genetic algorithms, with or without the use of parallel computing (Beasley and Chu, 1997; Liu and Wang, 2015), differential evolution improved with various local search techniques (using shift, swap and k-moves) (Sethanan and Pitakaso, 2016), as well as a path relinking method coupled with an ejection chain mechanism (Yagiura et al., 2006a), which

actually appears to be the most efficient approach for large instances, followed by tabu search algorithms.

Second, the *p-Median Problem* (pMP) is related to the WHDP. In the pMP, p facilities have to be chosen among a list of uncapacitated facilities, and the demands have to be assigned to the chosen facilities. A similar problem can be seen in the *Optimal Clustering Problem* (OCP), where p groups of demands have to be created, but there is no location to choose. Instead, the groups have to be made in a way that in each group, the objects have a maximum number of similarities (or for the pMP, the objects of each group are as close as possible, according to the Euclidean distance), and the groups are as different as possible. Here, the problem is about trying to group the demands into a fixed number of references while minimizing the give-away. Hence, considering that the give-away is a cost that increases with the number of different modules the demands have in a group (i.e., the less similar the demands are in a group, the higher the give-away will be), this is basically the same thing as solving the OCP, even if the costs considered are different. The pMP has only been solved exactly for instances with up to 500 clients and 50 facilities (Hansen and Mladenović, 1997), which is again far from the needs of large car manufacturers. The first specialized heuristic for the pMP is the c-means algorithm (Hansen and Mladenović, 2001a), where the neighborhood structure is defined by centroid-to-entity relocations followed by corresponding reassignments. This heuristic was also combined with local search procedures (Hatamlou, 2012). VNS methods were also developed for the pMP, and very favorably compared to different tabu search approaches (Hansen and Mladenović, 1997) (the worst solutions found with VNS being better than the best ones found with tabu search). Finally, evolutionary approaches have also been proposed for the pMP: a genetic algorithm that searches into the pre-generated assignments rather than constructing new ones (Cowgill et al., 1999), or differential evolution combined with the c-means algorithm (Tvrdík and Kivý, 2015).

Third, the WHDP can be reduced to a *Set Covering Problem* (SCP). Indeed, in the SCP, a population E of elements and a collection $(\Delta_i) \subset E$ (with $i \in \{1, \ldots, n\}$) of subsets are given, such that $\bigcup_{1 \leq i \leq n_i} \Delta_i = E$. The problem is to construct a set $C = \{\Delta_i \subset E \mid \bigcup \Delta_i = E\}$ while minimizing the costs. In the WHDP, that would be extracting a set $\mathcal{S}$ of solution references from a set $\mathcal{R}$ of possible references that minimizes the total give-away. The SCP can be solved exactly for instance with up to $|E| = 400$ and $n = 4,000$ (Yagiura et al., 2006b). It is again far from the data of the Company, which have thousands of demands and where $\mathcal{R}$ can contain more than 2^{30} elements. Various solution methods have been developed for the SCP: Lagrangian heuristics performing primal and dual relaxations and using subgradient optimization to improve the obtained bounds (Beasley, 1990; Ceria et al., 1995), local search methods employing 1-*opt* or 3-*flip* moves (Vasko and Wilson, 1984; Yagiura et al., 2006b), tabu search enhanced with a row-weighting scheme (Gao et al., 2015), simulated annealing enhanced with a morphing procedure that aims to reduce the cost and coverage correlation (Brusco et al., 1999), and both tabu search and simulated annealing combined

together (Jacobs and Brusco, 1995). Lagrangian heuristics have a good short-term behavior (i.e., they quickly converge to a local optimum), but they are outperformed by local search methods after a few minutes of computation. Finally, one can also find genetic algorithms with a specific feasibility operator (Beasley and Chu, 1996), and ant colony optimization using the dual values of the Lagrangian dual problem as heuristic estimates, and therefore offering a row-oriented approach (Al-Shihabi et al., 2015; Ren et al., 2010).

Finally, the WHDP belongs to the family of the *Component Commonality Problem* (CCP), which is a production problem that aims to use the same version of components across multiple products, instead of one version per product. Indeed, Menezes et al. (2016) show that the CCP is equivalent to the pMP, and propose different descent-greedy heuristics and a matheuristic to solve the problem (but only up to 500 demands and 12 modules). Considering different types of costs, a similar but small-sized problem (up to 100 demands, with only 8 possible modules) has been studied in Thonemann and Brandeau (2000). It was solved with a branch-and-bound algorithm for small instances, and with the help of simulated annealing for larger instances. Another small-sized subproblem (up to roughly 5,000 demands) has also been studied in Briant and Naddef (2004), and solved using Lagrangian relaxations and variable fixing. However, the produced wire harnesses were not created but chosen among the required ones, which makes the problem much simpler. The same subproblem has been explored in Agra et al. (2013), this time treating up to 50,000 demands with a decomposition method, and solved by simulated annealing and genetic algorithms. Here again, the references were simply chosen among the demands. Different papers (e.g., Desai, 2001; Hillier, 2002) explain that producing common components makes them more expensive and covers the demand less accurately, but it reduces manufacturing, storage and shipment costs. In addition, it decreases the risk in a stochastic demand environment by pooling volumes, hence a right balance has to be found. Actually, the WHDP can be connected with various production problems, for instance, which options to add to computers or phones, which tools to mount for production machines, where to drill the screw holes for pieces of ready-to-assemble furniture. This issue is fundamental when moving from low to high volumes markets (Jans, 2008). Finally, note that the component commonality feature has been studied for various applications in automotive production, like braking systems (e.g., Fisher et al., 1999; Ramdas et al., 2003).

In summary, various metaheuristics and particularly local search methods, have been used for problems related to the WHDP. Exact methods have been proposed for problems connected to that faced by car manufacturers, but not for instances as those of the Company, unless the set of possible references has been sufficiently reduced. These observations motivate the design of local search algorithms, and related metaheuristics such as variable neighborhood search or tabu search.

4. Integer linear model

In this section, an integer linear programming formulation of the problem is proposed. Next, a decomposition method, used in all the proposed algorithms, is presented. It simplifies the problem and reduces the size of the instances. As it will be used several times, the linearization of the boolean operators "OR" and "AND" (represented respectively by the symbols $\vee$ and $\wedge$) is detailed here. Let a and b be two boolean variables. The expression $a \wedge b$ (resp. $a \vee b$) can be replaced by another boolean variable c subject to the three constraints $c \leq a$, $c \leq b$, and $c \geq a + b - 1$ (resp. $c \geq a$, $c \geq b$, and $c \leq a + b$).

The following notation is used to state the mathematical models:

- $\mathcal{C}^E$ (resp. $\mathcal{C}^I$): set of the exclusion (resp. implication) constraints;
- $D_{md} = 1$ if the module $m \in \mathcal{M}$ is required in the demand $d \in \mathcal{D}$, $D_{md} = 0$ otherwise;
- $E_{me} = 1$ if the module $m \in \mathcal{M}$ is in the envelope $e \in \mathcal{E}$, $E_{me} = 0$ otherwise.

4.1. Mathematical formulation of the WHDP

Three types of decision variables are used in the model. First, $x_{dr} = 1$ if demand $d \in \mathcal{D}$ is covered by reference $r \in \mathcal{S}$, and $x_{dr} = 0$ otherwise. Second, $y_{er} = 1$ if envelope $e \in \mathcal{E}$ covers reference $r \in \mathcal{S}$, and $y_{er} = 0$ otherwise. Third, $z_{mr} = 1$ if module $m \in \mathcal{M}$ is in reference $r \in \mathcal{S}$, and $z_{mr} = 0$ otherwise. For each reference $r \in [1, N]$, the vector $(z_{mr})_{m \in \mathcal{M}}$ is denoted by z_r, and the presence cost of its modules is denoted by $c_r = f^C(z_r)$, where f^C is a non-linear but linearizable function.

The function f^C is not linear because some modules have a different cost if they are together or not in a reference. For example, there could be a cost c_1 for a module m_1, a cost c_2 for a module m_2, but a cost $c_{1\wedge 2} \neq c_1 + c_2$ if both m_1 and m_2 are in the reference (this is generalizable to more than two modules). The presence cost of these modules in a reference r is $c_1 \cdot z_{1r} \cdot (1 - z_{2r}) + c_2 \cdot z_{2r} \cdot (1 - z_{1r}) + c_{1\wedge 2} \cdot z_{1r} \wedge z_{2r}$, which is linearizable, because $z_{1r} \wedge z_{2r}$ is linearizable. Also, some groups of modules have a fixed cost if one or more modules of this group are in a reference. For example, there could be a cost $c_{1\vee 2}$ if at least one of the two modules m_1 and m_2 is in the reference (this is generalizable to more than two modules). The presence cost of these modules in a reference r is $c_{1\vee 2} \cdot z_{1r} \vee z_{2r}$, which is linearizable, because $z_{1r} \vee z_{2r}$ is linearizable.

The give-away of a demand d (i.e., the cost added by the unrequired modules that are in the reference r covering d) is: $Q_d \cdot \left[f^C(z_r) - f^C(D_d)\right]$, where D_d is the vector $(D_{md})_{m \in \mathcal{M}}$. Hence the total give-away is

$\sum_{dr} x_{dr} \cdot Q_d \cdot \left[f^C(z_r) - f^C(D_d)\right] = \left[\sum_{dr} x_{dr} \cdot Q_d \cdot f^C(z_r)\right] - \left[\sum_{dr} x_{dr} \cdot Q_d \cdot f^C(D_d)\right]$. Considering that the second term is the unavoidable constant cost of the required modules in each demand, the give-away of a demand d covered by a reference r can be reduced to $Q_d \cdot c_r$. The resulting objective function f to be minimized is

$$\sum_{dr} x_{dr} \cdot Q_d \cdot c_r. \tag{1}$$

The decisions are subject to implication and exclusion constraints. For each $r \in \mathcal{S}$, these are given under the following form: $g^I(z_r) \leq h^I(z_r)$ for the implication constraints, and $g^E(z_r) + h^E(z_r) \leq 1$ for the exclusion ones, where the functions g and h are non-linear functions that return 1 if the reference satisfies the corresponding criterion of the considered constraint, and 0 otherwise. More precisely, these functions have the form $(z_{m_{(11)}r} \wedge \cdots \wedge z_{m_{(1q_1)}r}) \vee \cdots \vee (z_{m_{(p1)}r} \wedge \cdots \wedge z_{m_{(pq_p)}r})$. That type of function can be linearized given that the operators "OR" and "AND" can be linearized, as explained before. As a consequence, all the implication and exclusion constraints are linearizable.

The constraints are the following:

$$\sum_r x_{dr} = 1 \qquad \forall d \in \mathcal{D} \tag{2}$$

$$\sum_e y_{er} = 1 \qquad \forall r \in [1, N] \tag{3}$$

$$x_{dr} \cdot D_{md} \leq z_{mr} \qquad \forall (m, d, r) \in \mathcal{M} \times \mathcal{D} \times [1, N] \tag{4}$$

$$y_{er} \cdot z_{mr} \leq E_{me} \qquad \forall (m, e, r) \in \mathcal{M} \times \mathcal{E} \times [1, N] \tag{5}$$

$$c_r = f^C(z_r) \qquad \forall r \in [1, N] \tag{6}$$

$$g^I_k(z_r) \leq h^I_k(z_r) \qquad \forall (r, k) \in [1, N] \times \mathcal{C}^I \tag{7}$$

$$g^E_k(z_r) + h^E_k(z_r) \leq 1 \qquad \forall (r, k) \in [1, N] \times \mathcal{C}^E \tag{8}$$

$$x_{dr} \in \{0, 1\} \qquad \forall (d, r) \in \mathcal{D} \times [1, N] \tag{9}$$

$$y_{er} \in \{0, 1\} \qquad \forall (e, r) \in \mathcal{E} \times [1, N] \tag{10}$$

$$z_{mr} \in \{0, 1\} \qquad \forall (m, r) \in \mathcal{M} \times [1, N]\,. \tag{11}$$

A solution $\mathcal{S}$ of the problem (P) defined by (1)-(11) consists in the N references chosen. Constraints (2) (resp. (3)) ensure that every demand (resp. reference) is covered by one reference (resp. envelope). Constraints (4) (resp. (5)) make sure that if a reference (resp. envelope) covers a demand (resp. reference),

the required modules in the demand (resp. reference) are also in the reference (resp. envelope). Constraints (6) calculate the presence cost of the modules, for each reference. Constraints (5) and (6) are non-linear, but as explained before, they can easily be linearized. Constraints (7) (resp. (8)) are the implication (resp. exclusion) constraints. These functions use the binary operators "AND" and "OR". They are computed with a sum of products of z_{mr}, and return 0 if the sum is 0, and 1 otherwise. A constraint $x_{11} = 1$ will be added to remove the symmetry of the model: it allows reducing the search space by removing solutions that are only different by a permutation of the references.

4.2. Decomposition of formulation (P)

As already discussed in the introduction, for each demand $d \in \mathcal{D}$, one and only one envelope $e \in \mathcal{E}$ covers d. It means that a reference r cannot cover two demands that are not covered by the same envelope. For every envelope $e \in \mathcal{E}$, $\mathcal{D}_e = \{d \in \mathcal{D} \mid e \text{ covers } d\}$. As a consequence, (P) can be decomposed into subproblems (P_e) that are the same as (P), but restricted to $\mathcal{D}_e$ instead of the whole set $\mathcal{D}$. The objective function of a problem (P_e) is then the same as the one of problem (P), but restricted to the demand covered by e.

As all the demands in $\mathcal{D}_e$ are covered by e and only e, we already know that all the references built for (P_e) will be covered by e as well. Hence, constraints (3) and (5) are useless in (P_e), like the variables y_{er} and, consequently, constraints (10). All the other constraints of (P) remain in (P_e). Also, the number of references imposed for (P_e) will be denoted N_e, with $N_e < N$. The different subproblems will be linked by the constraint $\sum_e N_e = N$. If a solution $\mathcal{S}_e$ is found for each (P_e), a solution $\mathcal{S}$ of (P) is therefore $\bigcup_e \mathcal{S}_e$.

5. Solution methods

We have first tried to solve formulation (P) with CPLEX, but it only managed to solve the problem for the smallest instance (with around 100 demands), and just for the three smallest number of references wanted (among around 40 possible values). This justifies once more the use of heuristic and metaheuristic methods to tackle this problem.

In this section, various solutions methods are proposed for the WHDP, ranging from basic constructive heuristics ($Gr^{\mathcal{D}}$ and $Gr^{\mathcal{E}}$, see subsection 5.1) and a descent search method (*DSM*, see subsection 5.2) to a more sophisticated metaheuristic based on variable neighborhood search (Hansen and Mladenović, 2001b) (*VNS*, see subsection 5.3). Note that tabu search (Glover, 1998) was also tested but did not provide conclusive improvements over *DSM*. For this reason, it will not be discussed in this paper.

5.1. Constructive Methods

Two different greedy algorithms have been implemented. The first starts from the demands and creates the references by sequentially grouping them (two at a time). The second starts from the envelopes and creates the references by sequentially separating the corresponding demands. Both heuristics follow the same pattern, described in Algorithm 1, which is also the core algorithm of all the other proposed solution methods. The main steps will be detailed later, as various approaches are possible.

Algorithm 1 Overview of the General Solution Approach

Initialization (for each $e \in \mathcal{E}$**)**

Demand / envelop initialization.

While the number N of references is not reached, **do**:

1. Select the envelope e with the optimal (i.e., minimal if group, maximal if separation) cost f_e.
2. Group / separate the corresponding references.
3. Update the grouping / separation costs affected by the last decision.
4. Perform a local search (e.g., *DSM*, *VNS*) to improve the solution (optional step).
5. If the total number $\sum_e N_e$ of references within all the subproblems has reached N, then stop.

5.1.1. Demands Greedy Algorithm $Gr^{\mathcal{D}}$

The *Demands Greedy Algorithm* ($Gr^{\mathcal{D}}$) is a bottom-up approach that starts from the demands as initial references, and groups the references two by two according to the cheapest grouping cost until the number N of references is reached. Note that grouping two references actually corresponds to merging two groups of demands into a new one, in order to create a new reference covering this new group of demands. The grouping cost $f^g(r, r')$ is the increase in the give-away if two references r and r' are grouped together. It can be computed as $f^g(r, r') = \sum_{d \prec r} Q_d \cdot (c_{r \cup r'} - c_r) + \sum_{d \prec r'} Q_d \cdot (c_{r \cup r'} - c_{r'})$, where $r \cup r'$ is the reference composed by the modules of both r and r'. The demand initialization is performed as in Algorithm 2, for each (P_e).

Algorithm 2 Demand Initialization for (P_e)

1. Set $\mathcal{S}_e = \mathcal{D}_e$.
2. For each possible couple $(r, r') \in \mathcal{S}_e \times \mathcal{S}_e$, compute $f^g(r, r')$.
3. Find the couple $(r_e, r'_e) \in \mathcal{S}_e \times \mathcal{S}_e$ that minimizes $f^g(r, r')$.
4. Set $f_e = f^g(r_e, r'_e)$.

In each step of $Gr^{\mathcal{D}}$ (step 1 of Algorithm 1), the smallest f_e among all the (P_e)s is found, and the corresponding references r_e and r'_e are grouped, which means they are replaced by $r_e \cup r'_e$ in the associated $\mathcal{S}_e$ (and hence all the demands covered by r or r' are now covered by $r \cup r'$). Next (step 2 of Algorithm 1), the costs are updated by computing, for each $r \in \mathcal{S}_e$, the cost $f^g(r, r_e \cup r'_e)$, and the process is restarted. If the number of references has reached N (step 4 of Algorithm 1), then the algorithm stops.

5.1.2. Envelopes Greedy Algorithm $Gr^{\mathcal{E}}$

The *Envelopes Greedy Algorithm* ($Gr^{\mathcal{E}}$) is a top-down approach that starts from the envelopes and creates the references by separating the corresponding demands in two parts, according to the best separation cost (ties are broken randomly) until the number N of references is reached. The separation cost $f^s(r, m)$ is the reduction of the give-away if a reference r containing a module m is split into two references r_m and $r_{\bar{m}}$ such that r_m covers each demand (covered by r) that contains m, and $r_{\bar{m}}$ covers each demand (covered by r) that does not contain m. It can be computed as $f^s(r, m) = \sum_{d \prec r | m \in d} Q_d \cdot (c_r - c_{r_m}) + \sum_{d \prec r | m \notin d} Q_d \cdot (c_r - c_{r_{\bar{m}}})$. The envelope initialization is performed as in Algorithm 3, for each (P_e).

Algorithm 3 Envelope Initialization

1. Set $\mathcal{S}_e = \{e\}$.
2. For each $r \in \mathcal{S}_e$ and $m \in \mathcal{M}$, compute $f^s(r, m)$.
3. Find the couple $(r_e, m_e) \in \mathcal{S} \times \mathcal{M}$ that maximizes $f^s(r, m)$.
4. Set $f_e = f^s(r_e, m_e)$.

In order to illustrate the separation process, consider an example with three demands d_1, d_2 and d_3, respectively composed of modules $\{m_1, m_2, m_4\}$, $\{m_1, m_2, m_6\}$, and $\{m_3, m_5, m_6\}$. A separation according to m_1 would be $r_{m_1} = d_1 \cup d_2 = \{m_1, m_2, m_4, m_6\}$ and $r_{\bar{m_1}} = d_4 = \{m_3, m_5, m_6\}$.

In each step of $Gr^{\mathcal{E}}$ (step 1 of Algorithm 1), the largest f_e among all the (P_e)s is found, and the corresponding reference r_e is separated according to module m_e, which means that it is replaced by r_{m_e} and $r_{\bar{m_e}}$ in the associated $\mathcal{S}_e$ (and hence all the demands covered by r are now covered by r_{m_e} if they contain m_e, and by $r_{\bar{m_e}}$ otherwise). Next (step 2), the costs are updated by computing, for each $m \in \mathcal{M}$, the costs $f^s(r_{m_e} m)$ and $f^s(r_{\bar{m_e}} m)$, and the process is restarted. If the number of references has reached N (step 4), then the algorithm stops.

In order to explore more solutions, using a less deterministic constructive algorithm, a *Greedy Randomized Procedure* was developed, inspired from the *Greedy Randomized Adaptive Search Procedure* (GRASP) (Feo

and Resende, 1995). Instead of making the best possible decision at each iteration, it chooses one randomly among the p best ones, where p is a parameter. This approach was tested within $Gr^{\mathcal{E}}$ only, because many ties are already encountered in $Gr^{\mathcal{D}}$ (remember that ties are always broken randomly). Unfortunately, this randomized approach did not lead to better results (various values of p have been tested), thus we did not pursue it.

5.2. Descent Search Method (DSM)

DSM is one of the most basic local search techniques. Starting from an initial solution, a local search moves at each step from the current solution $\mathcal{S}$ to a neighbor solution $\mathcal{S}'$ by performing a slight modification (called a move) on $\mathcal{S}$. In *DSM*, the best move in the whole neighborhood (according to the considered objective function) is performed, and the process stops at the first local optimum. A technique to move away from local optima will be discussed in Section 5.3. In order to adapt a *DSM* to a problem, the most sensitive issue is to design a relevant neighborhood structure (i.e., the definition of a move).

For the considered problem, a classic *demand-shift* neighborhood was first tested. It first shifts one demand from a reference to another, and then it reconstructs the references accordingly (i.e., add or remove modules to/from the two references in order to make them cover exactly their assigned demands). This move was not efficient at all. Indeed, shifting only one demand is usually not sufficient to reduce the reference wire harness of any module, and therefore not sufficient to reduce its cost. Moreover, this neighborhood is very large (it has a size of $O(N^2 \cdot |\mathcal{D}|)$, and $|\mathcal{D}|$ can reach $50,000$), hence it takes a lot of time to explore. The profitability of this neighborhood is therefore really small.

For this reason, a more appropriate *module-shift* neighborhood is proposed. It consists in shifting one module from one reference to another (which gives a neighborhood size order of $O(N^2 \cdot |\mathcal{M}|)$). More precisely, one module m is selected in a reference r, and then all the demands covered by r and containing m are shifted to another reference r' that can contain m or not. Finally, both r and r' are reconstructed accordingly. For example, consider an instance with four demands d_1, d_2, d_3 and d_4, respectively composed of modules $\{m_1, m_2\}$, $\{m_1, m_3\}$, $\{m_3, m_4\}$ and $\{m_4, m_5\}$. Assume that the current solution is made of the following two references: r composed of $\{m_1, m_2, m_3, m_4\}$ that covers d_1, d_2 and d_3, and r' composed of $\{m_4, m_5\}$ that covers d_4. Shifting the module m_3 (contained in d_2 and d_3) results in the following references: r composed of $\{m_1, m_2\}$ that covers d_1, and r' composed of $\{m_1, m_3, m_4, m_5\}$ that covers d_2, d_3 and d_4.

As the overall problem is divided into different subproblems, and only one subproblem at each step 1 of Algorithm 1 is modified, *DSM* is applied only on this subproblem in step 3 of Algorithm 1. The two resulting

methods are denoted as *DSM-Gr*$^{\mathcal{D}}$ and *DSM-Gr*$^{\mathcal{E}}$. Note that in *DSM-Gr*$^{\mathcal{D}}$, *DSM* is not performed for each N (from the number of demands $|\mathcal{D}|$ down to the number of envelopes $|\mathcal{E}|$), because this would take way too much time (typically, some weeks of computing time). Indeed, $|\mathcal{D}|$ goes from 150 to 55,000 demands, depending on the instance. Instead of that, *DSM* is performed on every subproblem as soon as $Gr^{\mathcal{D}}$ reaches the largest possible N value, and then down to $|\mathcal{E}|$ as explained above.

5.3. Variable Neighborhood Search (VNS)

VNS (Hansen and Mladenović, 2001b) is a well-known metaheuristic that combines a local search method with the use of a collection of neighborhood structures, denoted here as $\mathcal{N}_1, \mathcal{N}_2, \ldots$. A pseudo-code is given in Algorithm 4. *VNS* is composed of three phases: (1) *Shake* generates a solution $\mathcal{S}'$ in the selected neighborhood, (2) *Improve* applies a local search to $\mathcal{S}'$ to hopefully improve its quality, (3)*Relocate* decides if the search has to be relocated (or not) to another region of the solution space. It is always the case if the incumbent solution $\mathcal{S}$ is improved. Otherwise, larger modifications are performed in the next shaking phase. Usually, the neighborhood structures are ranked from the least to the most transformative (in terms of the modification of the involved solution). The main idea is to move from a neighborhood $\mathcal{N}_i$ to the next $\mathcal{N}_{i+1}$ if the former did not allow escaping from the incumbent local optimum. In order to adapt *VNS* to (P), the different neighborhood structures $\mathcal{N}_i$ have to be defined, as well as the employed local search in step *Improve*. The latter is simply *DSM*.

Algorithm 4 *VNS*

Initialization

Construct an initial solution $\mathcal{S}$.
Set $i = 1$ and $c = 0$.

While $c \leq c_{max}$, **do**:

1. *Shake:* generate a solution $\mathcal{S}'$ in $\mathcal{N}_i(\mathcal{S})$.
2. *Improve:* apply a local search on $\mathcal{S}'$ to obtain $\mathcal{S}$".
3. *Relocate:* **If** $\mathcal{S}$" is better than $\mathcal{S}$, set $\mathcal{S} = \mathcal{S}$" and $i = 1$. **Else** set $i = i + 1$.
4. **If** $i \geq i_{max}$, set $i = 1$ and $c = c + 1$.

The proposed neighborhood structures for (P) are inspired from the *module-shift* move explained above. It can be summarized as follows: a reference r is separated according to a module m, into a reference r_1 that does not contain m and another reference r_2 that contains m, as in $Gr^{\mathcal{E}}$. Next, r_1 replaces r, and r_2 is grouped with another reference r', as in $Gr^{\mathcal{D}}$. This observation leads to the neighborhood $\mathcal{N}_i$ used in

VNS, where i is an integer. For each i, it consists in sequentially separating i references with the separation process of $Gr^{\mathcal{E}}$, and then sequentially grouping i couples of references with the grouping process of $Gr^{\mathcal{D}}$. As each possible separation is performed according to a module, we fixed (after some tests) the value of i_{max} to $|\mathcal{M}|/2$. Once this maximum is reached, we cycle back to the beginning with $i = 1$. After some tests, we fixed the maximum number of cycles to $c_{max} = 3$. After having tested different options (see below), it was decided that the i separations are sequentially performed at best among $k\%$ of the possible ones (where k is a parameter tuned to 10 after preliminary experiments), and the i groupings are sequentially performed at best (i.e., with the lowest grouping cost). It is important to mention that, in each neighborhood structure, it is forbidden to group again two references that were just separated.

For the sake of completeness, for each neighborhood structure $\mathcal{N}_i$, note that the following options were tested to separate or group the i references:

1. separate at best among the whole neighborhood (i.e., with the largest possible value of k);
2. separate at random (i.e., with the smallest possible value of k);
3. group only the separated references (but of course with different references);
4. group according to the best "regret-grouping-cost" (i.e., the one that has the largest difference between its best grouping cost and its second best grouping cost);
5. forbid regrouping any pair of "descendant" of the separated references.

Exactly as for *DSM*, *VNS* is applied only in step 3 of Algorithm 1 on the subproblem that is modified. As two greedy algorithms are proposed, the two resulting methods are denoted as *VNS*-$Gr^{\mathcal{D}}$ and *VNS*-$Gr^{\mathcal{E}}$.

6. Results

All the algorithms were coded with C++ under Linux, and run on 3.4 GHz Intel Quad-core i7 processor with 8 GB of DDR3 RAM. The results of the proposed algorithms are compared to the ones obtained by $Gr^{\mathcal{D}}$, as $Gr^{\mathcal{D}}$ follows the logic of the Company's current algorithm, with a few coding improvements to make it faster. The six following methods will be compared, first in terms of costs, second in terms of speed, and third in terms of robustness: $Gr^{\mathcal{D}}$, *DSM*-$Gr^{\mathcal{D}}$, *VNS*-$Gr^{\mathcal{D}}$, $Gr^{\mathcal{E}}$, *DSM*-$Gr^{\mathcal{E}}$, *VNS*-$Gr^{\mathcal{E}}$.

Cost reduction (see Table 6) is the first objective in this strategic problem. However, speed (see Table 7) is important in the design phase to exclude the worst solutions, and robustness (see Table 8) matters too as a solution method has to be reliable. Because of a non-disclosure agreement, results are not shown in terms

of real costs but in terms of savings per wire harness and per vehicle. Knowing that the ten biggest car manufacturers have each sold more than 3 million vehicles in 2015 (OICA.net, 2015), and that a car contains several different wire harnesses (e.g., dashboard, engine, ceiling, trunk, doors), this should give the reader a representative idea of the potentially large impact of the proposed solution methods (even if just 0.05€ are saved per wire harness).

Table 6 presents the different characteristics of the 20 considered instances denoted I1 to I20. These instances are representative of the Company's production in terms of volume and diversity (as it concerns several years of production). Column 2 (resp. 3) gives information on the exclusion (resp. inclusion) constraints. The left part indicates if there is any exclusion (resp. implication) constraint in the instance, and the right part indicates if there is any exclusion (resp. implication) constraint that is not already satisfied by all the envelopes (resp. demands). Indeed, as the proposed algorithms construct the references based on the demands and the envelopes, such constraints are the only ones they have to deal with. These two columns show that there is never any exclusion constraint left, and that there are implication constraints left for only five instances. However, the implication constraints are easily tackled by adding wires (if needed) to each new reference, and the possibly additional costs are taken into account each time a separation/grouping cost is computed. Moreover, the number of these constraints in an instance is always very small (less than five). As a consequence, such constraints are easily integrated and satisfied in all the proposed algorithms. The fourth column shows if there are non-linear costs (as explained in Section 2). But, as for the implication constraints, these costs are included in the separation/grouping costs, hence they are again easily captured by the proposed methods. Column 5 shows the investigated interval for the number N of references. The next three columns give the number $|\mathcal{E}|$ of envelopes, $|\mathcal{M}|$ modules, and $|\mathcal{D}|$ of demands, respectively. As the instance decomposition according to the envelopes is used in all the algorithms, $|\mathcal{E}|$ will not allow discriminating the designed algorithms. On the contrary, as it will be explained using the next tables, $|\mathcal{D}|$ is the main characteristic that influences the obtained results. This is why the instances are sorted by increasing numbers of demands. From the value of $|\mathcal{M}|$ one can estimate the number of possible references (which is $2^{|\mathcal{M}|} - 1$ minus the references that do not satisfy the constraints), but as it tends to grow with $|\mathcal{D}|$, and as its interval range is small (from 20 to 64), $|\mathcal{D}|$ is sufficient to capture the instance difficulty. For the sake of completeness, as an indicator for the reader, the last two columns show the number $\max(|\mathcal{D}_e|)$ of demands in the largest subproblem, and the total number $\sum Q_d$ of wire harnesses to produce. The former element shows how much the decomposition simplifies the problem, and the latter gives an idea of the economic impact of the cost reduction per wire harness.

Remember that $Gr^{\mathcal{D}}$ is considered here as the reference method, because it corresponds to the current practice of the Company. Table 7 presents the quality of the solutions obtained by each algorithm. More

Table 6: Characteristics of the instances

Instance	**Exclusions**	**Implications**	**Costs**	N	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{D}\rvert$	$\max(\lvert\mathcal{D}_e\rvert)$	$\sum Q_d$
I1	yes / no	yes / no	nonlin	[6, 40]	6	27	167	88	75,892
I2	no / no	no / no	lin	[30, 100]	30	46	542	68	107,101
I3	yes / no	yes / yes	nonlin	[10, 40]	10	32	1,157	596	166,100
I4	yes / no	yes / no	lin	[7, 40]	7	20	1,261	445	90,493
I5	no / no	no / no	lin	[36, 100]	36	38	1,696	168	201,747
I6	yes / no	no / no	nonlin	[24, 70]	24	30	1,700	153	41,803
I7	no / no	yes / no	lin	[7, 40]	7	35	3,320	912	214,323
I8	no / no	yes / no	nonlin	[6, 40]	6	33	3,847	1,868	200,040
I9	no / no	no / no	lin	[6, 40]	6	37	4,534	1,391	30,000
I10	yes / no	yes / yes	nonlin	[14, 40]	14	33	5,162	876	114,992
I11	no / no	yes / no	lin	[6, 40]	6	35	6,807	3,474	150,000
I12	no / no	yes / no	lin	[24, 70]	24	34	8,713	2,048	27,700
I13	yes / no	no / no	lin	[6, 40]	6	37	8,786	4,698	150,000
I14	no / no	yes / yes	lin	[8, 40]	8	46	11,638	4,602	100,000
I15	yes / no	yes / yes	lin	[32, 100]	32	64	22,059	1,786	29,899
I16	yes / no	yes / yes	lin	[15, 40]	15	52	23,967	3,527	150,015
I17	no / no	yes / no	lin	[6, 40]	6	54	30,559	20,947	49,992
I18	yes / no	yes / no	nonlin	[5, 40]	5	52	39,074	19,714	142,500
I19	no / no	no / no	lin	[10, 40]	10	51	44,152	15,661	86,000
I20	no / no	no / no	lin	[16, 40]	16	58	54,792	19,208	80,000

precisely, it presents the best (over 10 runs) average saving (in euros) on the give-away of one wire harness (averaged over all the possible number N of references) with respect to the solution value found by $Gr^{\mathcal{D}}$ (it is why the column of this algorithm contains only zeros). First, the results show that $Gr^{\mathcal{D}}$ clearly outperforms $Gr^{\mathcal{E}}$. This is not a surprise since $Gr^{\mathcal{D}}$ is more thorough. Indeed, it starts with all the individual demands and then groups them two by two (which makes $|\mathcal{D}| - N_{\min}$ steps) at best after exploring all the possible groups. In contrast, $Gr^{\mathcal{E}}$ starts with the groups defined by the envelope-covering and then separates them in two (which makes only $N_{\max} - N_{\min} \approx 4 \cdot |\mathcal{E}| - N_{\min}$ steps) after exploring the already presented separation options (exploring all the possible separations is not possible: for an envelope containing D demands, there are $2^D - 1$ possible separations). Second, the intensification ability provided by *DSM* leads to important savings. Indeed, on average, it improves $Gr^{\mathcal{E}}$ by 0.98€ per wire harness, and $Gr^{\mathcal{D}}$ by 0.39€ per wire harness. As already mentioned, we also experimented with a tabu search approach (instead of *DSM*). The presence of the module used for a separation of a reference was tabu in this reference for a few iterations (making the moved demands tabu was also tried, but unsuccessfully). Unfortunately, it does not perform better than *DSM*. Even if we do not provide the results obtained by tabu search, its disappointing behavior shows that it is difficult to escape from local optima. Indeed, the diversification mechanism of the tabu tenures is not sufficient to escape from such local optima. In contrast, the much stronger diversification capability of *VNS* leads to additional improvements, confirming the relevance of the proposed neighborhoods. To better

understand this, one can observe that *DSM* separates a reference in two and then groups one of the two wire harnesses obtained with another reference (in order to stay with N references), whereas *VNS* separates i times and then groups i times. It can thus correspond to big jumps in the solution space. On average, *VNS* saves (with respect to *DSM*) 0.19€ per wire harness if applied to $Gr^{\mathcal{E}}$, and 0.14€ if applied to $Gr^{\mathcal{D}}$. When compared to the current practice of the Company (i.e., $Gr^{\mathcal{D}}$), *VNS* saves on average 0.53€ per wire harness. At first sight, such savings might appear negligible. However, if one reasonably assumes that (1) there are several different wire harnesses per car and (2) millions of cars are sold every year (which is the case of many car producers), such savings corresponds to significant annual cost reductions.

Table 7: Quality: average savings of the solutions methods

Instance	$\|\mathcal{D}\|$	$Gr^{\mathcal{E}}$	$Gr^{\mathcal{D}}$	*DSM*-$Gr^{\mathcal{E}}$	*DSM*-$Gr^{\mathcal{D}}$	*VNS*-$Gr^{\mathcal{E}}$	*VNS*-$Gr^{\mathcal{D}}$
I1	167	-0.34	0.00	-0.27	0.00	-0.07	0.01
I2	542	-2.45	0.00	-1.51	0.04	-0.89	0.06
I3	1,157	-1.25	0.00	-0.01	0.06	0.13	0.14
I4	1,261	-0.48	0.00	-0.07	0.05	-0.06	0.08
I5	1,696	-0.71	0.00	-0.43	0.01	-0.35	0.03
I6	1,700	-0.03	0.00	0.06	0.08	0.08	0.12
I7	3,320	-0.26	0.00	0.15	0.16	0.24	0.26
I8	3,847	-1.38	0.00	-1.06	0.04	-0.63	0.07
I9	4,534	-0.07	0.00	0.06	0.05	0.06	0.08
I10	5,162	-0.58	0.00	0.07	0.10	0.10	0.15
I11	6,807	-1.11	0.00	-0.15	0.05	-0.09	0.15
I12	8,713	-0.02	0.00	0.00	0.00	0.00	0.01
I13	8,786	-1.08	0.00	-0.12	0.13	-0.01	0.18
I14	11,638	-0.42	0.00	0.12	0.24	0.14	0.28
I15	22,059	-0.50	0.00	0.65	0.43	0.74	0.72
I16	23,967	-1.09	0.00	0.30	0.27	0.60	0.63
I17	30,559	-0.61	0.00	0.04	0.16	0.20	0.24
I18	39,074	-0.34	0.00	0.51	0.46	0.60	0.60
I19	44,152	-0.02	0.00	3.74	3.62	4.36	4.29
I20	54,792	-2.85	0.00	2.00	1.75	2.63	2.52
Average		**-0.78**	**0.00**	**0.20**	**0.39**	**0.39**	**0.53**

Table 8 shows the computing times (in seconds) of the different methods. These times are averaged over 10 runs, but not over all the possible values of N. Indeed, as the algorithms are incremental, the value of a solution with N references is based on the solution found with $N+1$ references if $Gr^{\mathcal{D}}$ is used, but $N-1$ references if $Gr^{\mathcal{E}}$ is used. This table shows first that $Gr^{\mathcal{E}}$ is much faster than $Gr^{\mathcal{D}}$: looking at the largest instance, it is even 200 times faster. This is an expected observation, as the number of steps (groups/separations) performed by $Gr^{\mathcal{E}}$ is way smaller than that performed by $Gr^{\mathcal{D}}$, as explained previously. Next, *DSM* does not affect too much the computing times. It seems to confirm the above conjecture on the attractiveness of the local optima. Indeed, *DSM* is likely to be quickly trapped. *DSM* slows down $Gr^{\mathcal{E}}$

(which is straightforward, as the solutions found without *DSM* were clearly not optimal, see Table 7), but it actually sometimes speeds up $Gr^{\mathcal{D}}$. This is possible because the solutions found by $Gr^{\mathcal{D}}$ are better than the ones of $Gr^{\mathcal{E}}$. Therefore, the descent cannot go far away from the input solution, and if *DSM* can provide an interesting improvement, it can help moving to a significantly smaller subproblem, resulting in time savings when compared to $Gr^{\mathcal{D}}$ only. Finally, *VNS* slows down both algorithm drastically: with $Gr^{\mathcal{D}}$, it can take up to 12 hours, and with $Gr^{\mathcal{E}}$, up to 18 hours. *VNS* takes more time with $Gr^{\mathcal{E}}$ because the solutions provided by $Gr^{\mathcal{E}}$ are poorer local optima than those obtained by $Gr^{\mathcal{D}}$ (as already discussed). Hence, the path down to the stronger local optima is longer. As this problem is a strategic one, the computing time is not an issue for the Company, as long as there is a cost reduction. Nevertheless, the existence of an extremely quick algorithm that still gives accurate solutions (as $Gr^{\mathcal{E}}$ that always takes less than a minute) has its interest for the Company: it can help during the initial phase, when designing possible solutions for the car models. Indeed, $Gr^{\mathcal{E}}$ has the obvious advantage of quickly providing an accurate and relevant (from a practical standpoint) upper bound on the expected costs. In a second phase, for the remaining choices (among the designed car models), *VNS* is of course the recommended method to further reduce the costs.

Table 8: Speed: average computing times (in seconds) of the solution methods

Instance	$\|\mathcal{D}\|$	$Gr^{\mathcal{E}}$	*DSM*-$Gr^{\mathcal{E}}$	*DSM*-$Gr^{\mathcal{D}}$	$Gr^{\mathcal{D}}$	*VNS*-$Gr^{\mathcal{D}}$	*VNS*-$Gr^{\mathcal{E}}$
I1	167	0	0	0	0	9	13
I2	542	0	0	0	0	26	40
I3	1,157	1	8	7	4	550	947
I4	1,261	0	1	2	1	31	49
I5	1,696	0	1	1	0	79	103
I6	1,700	1	2	3	1	102	188
I7	3,320	1	7	11	7	324	531
I8	3,847	1	7	16	11	436	745
I9	4,534	1	7	14	11	521	964
I10	5,162	2	12	19	11	454	748
I11	6,807	2	23	47	48	1,614	2,530
I12	8,713	1	4	18	14	283	452
I13	8,786	14	77	252	158	7,234	11,004
I14	11,638	7	36	79	86	2,463	4,315
I15	22,059	17	122	366	214	13,896	18,094
I16	23,967	15	97	243	208	3,202	7,192
I17	30,559	30	162	3,115	8,488	31,227	43,341
I18	39,074	53	350	4,302	4,633	40,425	64,966
I19	44,152	30	226	2,624	2,901	26,556	38,271
I20	54,792	44	267	8,029	3,879	23,499	40,330
Average		**11**	**70**	**957**	**1,034**	**7,647**	**11,741**

Table 9 gives the average (over 10 runs) standard deviations σ (again, averaged over the possible numbers N of references) obtained for the average give-away. This table shows another advantage of $Gr^{\mathcal{E}}$: it is not only

the quickest algorithm, but also the more robust (its costs are completely stable), which is not the case of $Gr^{\mathcal{D}}$, which has on average a standard deviation of 0.04€. This table also shows that *DSM* does not change the robustness of $Gr^{\mathcal{E}}$, thus *DSM*-$Gr^{\mathcal{E}}$ shows a real interest for the Company: it gives better solutions than $Gr^{\mathcal{D}}$, it is up to a hundred times faster, and with no variability. In other words, it is a very efficient and reliable tool to use in various contexts encountered by the decision maker.

Table 9: Robustness: average standard deviations of the solution methods

Instance	$\|\mathcal{D}\|$	$Gr^{\mathcal{E}}$	*DSM*-$Gr^{\mathcal{E}}$	*VNS*-$Gr^{\mathcal{D}}$	*DSM*-$Gr^{\mathcal{D}}$	$Gr^{\mathcal{D}}$	*VNS*-$Gr^{\mathcal{E}}$
I1	167	0.00	0.00	0.00	0.00	0.00	0.05
I2	542	0.00	0.00	0.00	0.00	0.00	0.01
I3	1,157	0.00	0.00	0.01	0.00	0.00	0.01
I4	1,261	0.00	0.00	0.00	0.00	0.00	0.01
I5	1,696	0.00	0.00	0.01	0.00	0.00	0.01
I6	1,700	0.00	0.00	0.00	0.00	0.00	0.00
I7	3,320	0.00	0.00	0.03	0.04	0.03	0.02
I8	3,847	0.00	0.00	0.02	0.00	0.00	0.15
I9	4,534	0.00	0.00	0.00	0.00	0.01	0.01
I10	5,162	0.00	0.00	0.01	0.00	0.00	0.01
I11	6,807	0.00	0.00	0.02	0.00	0.01	0.01
I12	8,713	0.00	0.00	0.00	0.00	0.00	0.00
I13	8,786	0.00	0.00	0.02	0.03	0.03	0.04
I14	11,638	0.00	0.00	0.01	0.02	0.03	0.00
I15	22,059	0.00	0.00	0.02	0.06	0.08	0.01
I16	23,967	0.00	0.00	0.02	0.01	0.01	0.01
I17	30,559	0.00	0.00	0.01	0.03	0.07	0.02
I18	39,074	0.00	0.00	0.01	0.03	0.05	0.01
I19	44,152	0.00	0.00	0.13	0.22	0.21	0.16
I20	54,792	0.00	0.00	0.12	0.17	0.21	0.16
Average		**0.00**	**0.00**	**0.02**	**0.03**	**0.04**	**0.04**

Typical variation graphs (i.e., presenting the results according to the various values of N) obtained by all the algorithms for four instances (I4, I6, I16 and I19) are shown in Figures 1, 2, 3, 4. For each allowed number N of references, it shows the best value of the average give-away (averaged over all demands and over 10 runs). The scale of this value is concealed, due to a non-disclosure agreement. The same conclusions can be made with this graph: $Gr^{\mathcal{D}}$ outperforms $Gr^{\mathcal{E}}$, and they are both improved by *DSM*, and even more by *VNS*. One can also observe that the difference between the average give-away obtained by the different algorithms grows at the beginning and then decreases slowly. This is due to the fact that the give-away is the same for all the methods at the beginning with $N = |\mathcal{E}|$, and at the end with $N = |\mathcal{D}|$ (which is out of the scope of the graph, as the number N of references wanted by the Company should be less than $4 \cdot |\mathcal{E}|$). Also, as N grows, the chosen wire harnesses will cover fewer different demands, therefore the possible reductions become less and less important in terms of costs. These four graphs show the most representative cases that can occur

for all the instances: either there is no real difference between the six different algorithms (see instance I6), or there is no real difference between $Gr^{\mathcal{D}}$ and $Gr^{\mathcal{E}}$ but *DSM* improves both widely and *VNS* a bit more (see instance I19), or the solutions found by $Gr^{\mathcal{E}}$ are far from the ones found by $Gr^{\mathcal{D}}$ but then *DSM* fills up the gap and *VNS* improves solutions even more (see instances I4 and I16). In that last case, there are two different situations: in the first one (see instance I4), the solutions of $Gr^{\mathcal{D}}$ are better than the solutions of $Gr^{\mathcal{E}}$ for the small values of N, but this gap decreases quickly and the solutions tends to be closer for the big values of N. On the contrary (see instance I16), the solutions of $Gr^{\mathcal{D}}$ tend to be further away from the solutions of $Gr^{\mathcal{E}}$ as N grows. This can be explained by the number of different demands contained in each instance. Indeed, I4 has around a thousand demands whereas I16 has around 25 thousand demands: it means that for I4, N could take around a thousand different values, when in I16 it could take around 25 thousands different values. Therefore, the gap between $Gr^{\mathcal{D}}$ and $Gr^{\mathcal{E}}$ for I16 will decrease, but for values of N that we do not consider (as the Company is not interested in a huge number of references).

Figure 1: Variation graphs of the average give-away obtained for instance I4

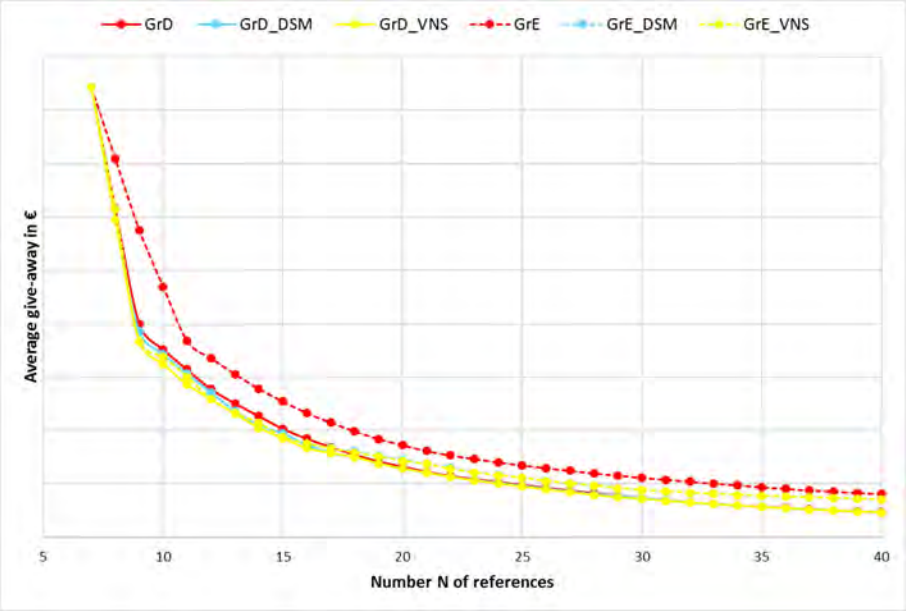


Figure 2: Variation graphs of the average give-away obtained for instance I6

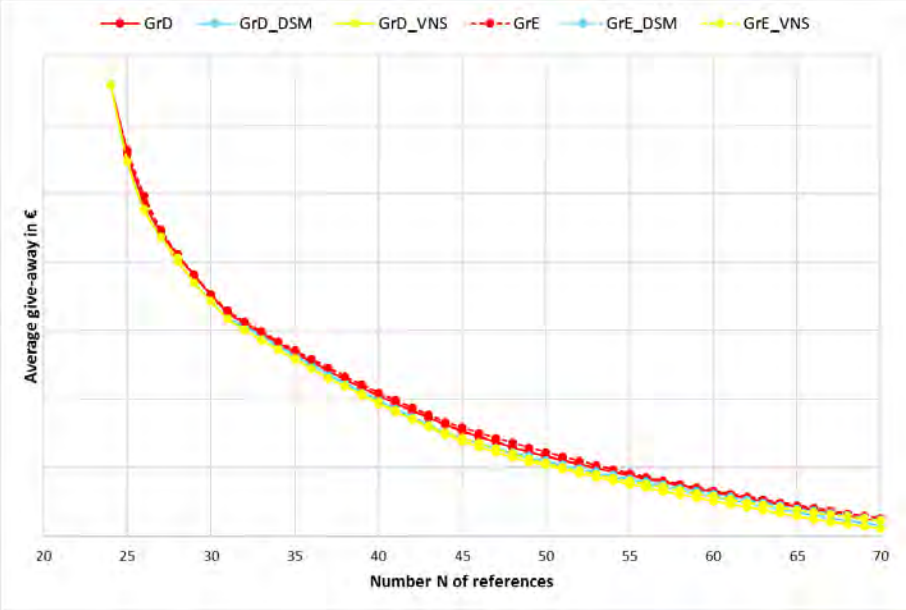


Figure 3: Variation graphs of the average give-away obtained for instance I16

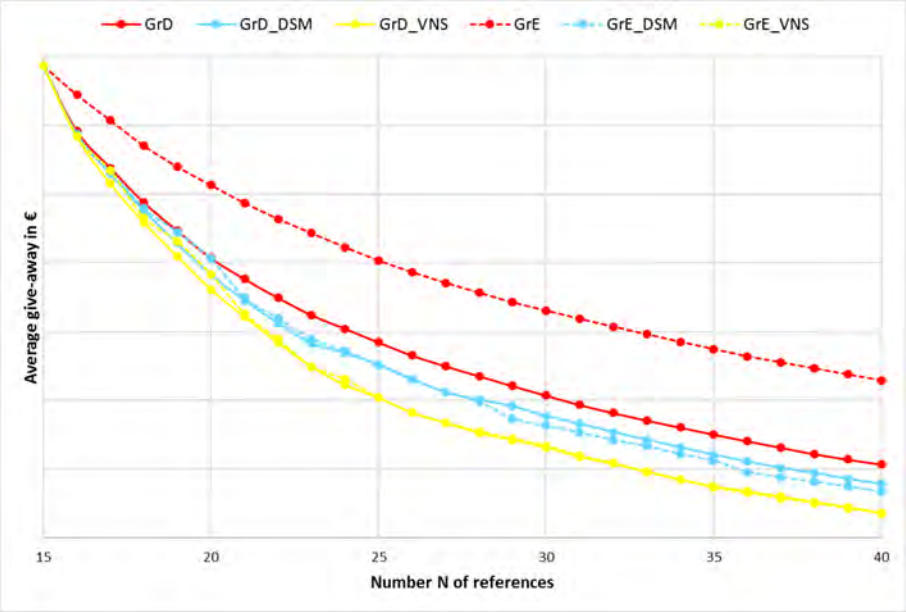
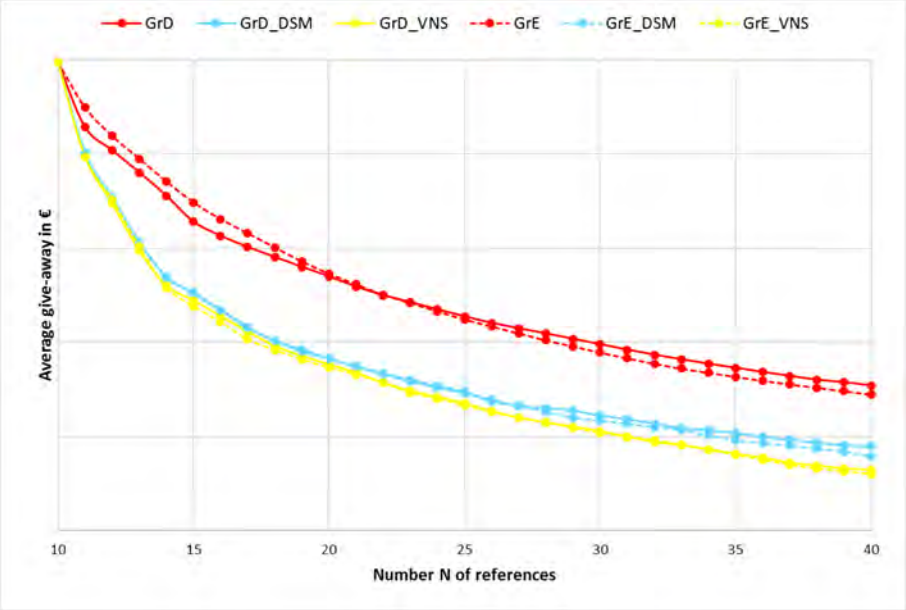


Figure 4: Variation graphs of the average give-away obtained for instance I19



Following the previous discussion, it is natural to compare the values obtained by the different algorithms for each value of N. Table 10 shows the average give-away obtained by each algorithm (averaged over all the demands, over 10 runs, and over all the instances), and for different values of N (more exactly, N chosen at $x\%$ of its possible interval, e.g., if N can be in $[10, 40]$, when the table shows N at 50%, it means that $N = 10 + 0.50 \times (40 - 10) = 25$). For $N = 0\%$, there is no gap between the algorithms, as the solution proposed are the set of envelopes. This table shows that, on average, the gap between $Gr^{\mathcal{D}}$ and $Gr^{\mathcal{E}}$ tends to decrease as N grows, but that *DSM* and *VNS* seems to improve the most the solutions in the middle of the possible interval (around 30% for *DSM* and around 50% for *VNS*). This is good news for the Company, as there is a significant probability that they will choose a value in the middle of the interval instead of at the ends: indeed, they do not want to only use the envelopes (which is the case at 0%), nor do they want to have a large number of references (which is the case at 100%).

7. Conclusion

The WHDP is a challenging combinatorial optimization problem faced by car manufacturers. Real instances tend to be very large, therefore ruling out the use of exact methods. A mathematical model was proposed, with a decomposition method to reduce the size of the problem. Next, two greedy heuristics, improved by a

Table 10: Variation of the average savings of the solutions methods (according to N).

N	$Gr^{\mathcal{E}}$	$Gr^{\mathcal{D}}$	DSM-$Gr^{\mathcal{E}}$	DSM-$Gr^{\mathcal{D}}$	VNS-$Gr^{\mathcal{E}}$	VNS-$Gr^{\mathcal{D}}$
0%	0.00	0.00	0.00	0.00	0.00	0.00
10%	-0.96	0.00	0.19	0.28	0.31	0.36
20%	-0.93	0.00	0.28	0.41	0.44	0.51
30%	-0.85	0.00	0.32	0.46	0.49	0.57
40%	-0.83	0.00	0.27	0.44	0.44	0.57
50%	-0.79	0.00	0.24	0.45	0.45	0.58
60%	-0.76	0.00	0.22	0.44	0.42	0.59
70%	-0.74	0.00	0.19	0.41	0.39	0.59
80%	-0.73	0.00	0.15	0.38	0.36	0.58
90%	-0.72	0.00	0.13	0.36	0.34	0.56
100%	-0.72	0.00	0.09	0.34	0.31	0.54
Average	-0.78	0.00	0.20	0.39	0.39	0.53

descent search method (*DSM*) and a variable neighborhood search (*VNS*), were designed. The first greedy algorithm $Gr^{\mathcal{D}}$, combined with *VNS*, provides the best solutions, but in a significant computing time (up to 18 hours), and with a small standard deviation. Therefore, it is recommended in a final decision phase. Otherwise, the second greedy algorithm $Gr^{\mathcal{E}}$, combined with *DSM*, provides comparable solutions, but much quicker (with a computing time only up to 6 minutes), and with a perfect robustness (i.e., zero standard deviation). Hence, it is recommended for an initial design phase. According to the results obtained on the Company's instances, these algorithms save respectively 0.53€ and 0.20€ per wire harness. Considering that a car has several wire harnesses, and that most car manufacturers sell millions of cars per year, small savings per wire harness correspond to significant annual cost reductions. It is important to mention here that our results were formally validated by the Company.

A possible extension of this work would be to develop an exact method using column generation, the columns being the references. Indeed, the structure of the problem makes the decomposition in a master problem and subproblems easy: the first one would be to create the best feasible solution using a known population of references, the second to create a new interesting reference. Because the total number of possible references is very large (equal to $2^{|\mathcal{M}|} - 1$), an exact method based on complete enumeration is not possible, but a column generation might prove effective.

References

A. Agra, J.O. Cerdeira, and C. Requejo. Using decomposition to improve greedy solutions of the optimal diversity management problem. *International Transactions in Operational Research*, 20(5):617–625, 2013.

S. Al-Shihabi, M. Arafeh, and M. Barghash. An improved hybrid algorithm for the set covering problem. *Computers & Industrial Engineering*, 85:328–334, 2015.

J.E. Beasley. A Lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37(1):151–164, 1990.

J.E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404, 1996.

J.E. Beasley and P.C. Chu. A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48(8):804–809, 1997.

O. Briant and D. Naddef. The Optimal Diversity Management Problem. *Operations Research*, 52(4):515–526, 2004.

M.J. Brusco, L.W. Jacobs, and G.M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems. *Annals of Operations Research*, 86(0): 611–627, 1999.

S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based Heuristic for Large-scale Set Covering Problems. *Program*, 81(2):1–15, 1995.

M.C. Cowgill, R.J. Harvey, and L.T. Watson. A genetic algorithm approach to cluster analysis. *Computers & Mathematics with Applications*, 37(99):99–108, 1999.

P. Desai. Product Differentiation and Commonality in Design: Balancing Revenue and Cost Drivers. *Management Science*, 47(1):37–51, 2001.

J. Diaz and E. Fernández. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132(1):22–38, 2001.

T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

M. Fisher, K. Ramdas, and K. Ulrich. Component Sharing in the Management of Product Variety : A Study of Automotive Braking Systems. *Management Science*, 45(3):297–315, 1999.

C. Gao, X. Yao, T. Weise, and J. Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.

M.R. Garey and D.S. Johnson. *Computers and Intractability*. W. H. Free edition, 1979.

M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer edition, 2010.

F. Glover. Tabu search: wellsprings and challenges. *European Journal of Operational Research*, 106(2-3): 221–225, 1998.

P. Hansen and N. Mladenović. Variable Neighborhood Search for the p-median. *Location Science*, 5(4): 207–226, 1997.

P. Hansen and N. Mladenović. J-Means: a new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34(2):405–413, 2001a.

P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001b.

A. Hatamlou. In search of optimal centroids on data clustering using a binary search algorithm. *Pattern Recognition Letters*, 33(13):1756–1760, 2012.

M.S. Hillier. The costs and benefits of commonality in assemble-to-order systems with a (Q,r)-policy for component replenishment. *European Journal of Operational Research*, 141(3):570–586, 2002.

L.W. Jacobs and M.J. Brusco. Note: A Local-Search Heuristic for Large Set-Covering Problems. *Naval Research Logistics*, 42(2):1129–1140, 1995.

T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810–826, 2009.

R. Jans. Analysis of an industrial component commonality problem. *European Journal of Operational Research*, 186(2):801–811, 2008.

V. Jeet and E. Kutanoglu. Lagrangian relaxation guided problem space search heuristics for generalized assignment problems. *European Journal of Operational Research*, 182(3):1039–1056, 2007.

E. Labro. The Cost Effects of Component Commonality: A Literature Review Through a Management-Accounting Lens. *Manufacturing & Service Operations Management*, 6(4):358–367, 2004.

M. Laguna, J.P. Kelly, J.L. González-Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82(1):176–189, 1995.

I. Litvinchev, M. Mata, S. Rangel, and J. Saucedo. Lagrangian heuristic for a class of the generalized assignment problems. *Computers and Mathematics with Applications*, 60(4):1115–1123, 2010.

Y.Y. Liu and S. Wang. A scalable parallel genetic algorithm for the Generalized Assignment Problem. *Parallel Computing*, 46:98–119, 2015.

E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

M. Menezes, D. Ruiz-hernández, and R. Guimaraes. The component commonality problem in a real multi-dimensional space: An algorithmic approach. *European Journal of Operational Research*, 249(1):105–116, 2016.

N. Mladenović, J. Brimberg, P. Hansen, and J.A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

OICA.net. Production statistics. `http://www.oica.net/category/production-statistics/2015-statistics/`, 2015.

K. Ramdas, F. Marshall, and K. Ulrich. Managing Variety for Assembled Products: Modeling Component Systems Sharing. *Manufacturing & Service Operations Management*, 5(2):142–156, 2003.

Z.-G. Ren, Z.-R. Feng, L.-J. Ke, and Z.-J. Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, 58(4):774–784, 2010.

H.E. Romeijn and D.R. Morales. A class of greedy algorithms for the generalized assignment problem. *Discrete Applied Mathematics*, 103(1-3):209–235, 2000.

K. Sethanan and R. Pitakaso. Improved differential evolution algorithms for solving generalized assignment problem. *Expert Systems with Applications*, 45:450–459, 2016.

Statista.com. Number of existing car models offered in the U.S. market from 1997 to 2016. `https://www.statista.com/statistics/200097/number-of-existing-car-models-on-the-us-market-since-1990/`, 2017.

U. Thonemann and M. Brandeau. Optimal Commonality in Component Design. *Operations Research*, 48 (1):1–19, 2000.

J. Tvrdík and I. Kivý. Hybrid differential evolution algorithm for optimal clustering. *Applied Soft Computing*, 35:502–512, 2015.

F.J. Vasko and G.R. Wilson. An efficient heuristic for large set covering problems. *Naval Research Logistics*, 31(1):163–171, 1984.

M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization*, 1(1):87–98, 2004.

M. Yagiura, T. Ibaraki, and F. Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169(2):548–569, 2006a.

M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172(2):472–499, 2006b.

N. Zufferey. Metaheuristics: Some Principles for an Efficient Design. *Computer Technology & Application*, 3(6):446–462, 2012.