Designing Spacecraft Command Loops Using Two-Dimension Vehicle Routing

Eliott Roynette[1,2(✉)], Bertrand Cabon[1], Cédric Pralet[2], and Vincent Vidal[2]

[1] Airbus DS Toulouse, Toulouse, France
{eliott.roynette,bertrand.cabon}@airbus.com
[2] ONERA – The French Aerospace Lab, 31055 Toulouse, France
{cedric.pralet,vincent.vidal}@onera.fr

Abstract. In the race to space, the reduction of the cost and weight of spacecrafts is one of the keys to progress further and further. In this perspective, a key component to consider in a spacecraft is the command system. The latter is a vital system which gives control over all the onboard devices, and which is materialized as a set of cables connecting devices with a set of controllers. To reduce the cost of development and the weight of this command system, there is a need to define optimization techniques for helping space engineers during the design phase. The objective of this paper is to present the problem tackled, which is a kind of Vehicle Routing Problem which we call a Two-dimension Vehicle Routing Problem, and to compare several solution techniques. One of these techniques is currently used for production.

1 Problem Description

In this paper, we tackle a problem from the space domain where the goal is to design the architecture responsible for controlling devices onboard spacecrafts. Spacecrafts typically contain hundreds to thousands of separate devices, which allow the mission assigned to the spacecrafts to be achieved.

A first possible way for being able to send commands to each of these devices during flight would be to define one command loop per device, linking the device with the central controller of the spacecraft. The main drawback of this approach would be a huge increase in weight, because physically each command loop is materialized as a cable. This is why space engineers developed command architectures where command loops are shared between devices. More precisely, the spacecrafts we consider use architectural components called *command matrices*, which are illustrated in Fig. 1(a). At an abstract level, a command matrix is a component made of rows and columns. One command loop is associated with each row and one command loop is associated with each column. Then, in the command architecture, each device d is placed in one row r and one column c of one matrix, and for sending a command signal to d, it suffices to send one signal on the command loop associated with r and one signal on the command loop associated with c. Doing so, the two emitted signals simultaneously reach device d positioned at the intersection between r and c. Several matrices are used

C.-G. Quimper (Ed.): CPAIOR 2016, LNCS 9676, pp. 303–318, 2016.
DOI: 10.1007/978-3-319-33954-2_22

instead of a single one mainly due to power and resistance constraints (placing too many devices on a single loop would generate too much resistance, reducing or destroying the functionality of the circuit). On real spacecrafts, command matrices typically range from size 2×2 to size 8×16 or even 16×16.

The main advantage of this command matrix architecture is that command loops, and therefore cables, are shared between devices. This reduces the weight and cost of the spacecraft. However, one difficulty is that for spacecraft reliability issues, some *segregation constraints* between devices must be satisfied, meaning that some devices are not allowed to be placed on the same row or on the same column of a command matrix. For instance, spacecrafts are often composed of sets of redundant devices, and the latter must not share a common command loop so that in case of failure of one loop, at least one device is still available. As a result, designing the placement of devices inside command matrices quickly becomes a combinatorial task.

Another feature of such an architecture is that each row (resp. each column) of a command matrix only defines the set of devices present on the command loop associated with that row (resp. with that column). On this point, one must also decide on the order of traversal of these devices by a physical cable, with the objective of minimizing the required cable length. For example, in Fig. 1(b), devices d_4, d_3, d_7, d_8 placed in row 2 of command matrix 1 are traversed in order $[d_3, d_7, d_8, d_4]$ in the physical command loop. Similarly, devices d_2, d_6, d_9 placed

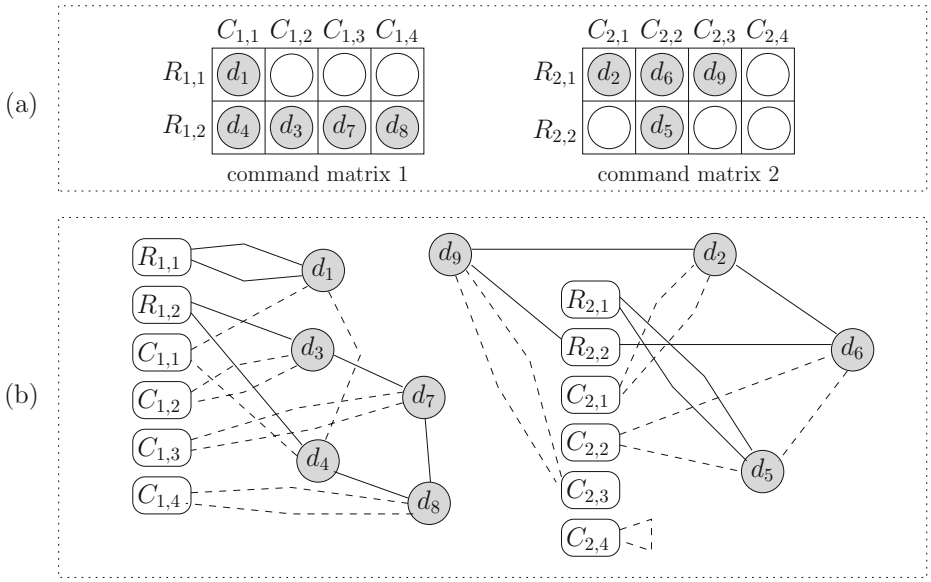


Fig. 1. Architecture for commanding devices: (a) allocation of positions in command matrices to a set of devices numbered from d_1 to d_9; (b) command loops physically used onboard the spacecraft; command loops associated with rows and columns are depicted using solid lines and dashed lines respectively

in row 1 of command matrix 2 are traversed in order $[d_9, d_2, d_6]$ by the cable associated with the command loop of this row.

The design problem considered in this paper consists in placing devices in command matrices and in deciding on the order in which these devices are traversed by cables, while satisfying segregation constraints and minimizing the total length (or weight) of cables used. In other words, the goal is to produce solutions such as the one depicted in Fig. 1. Also, in the problem we consider, the physical placement (x, y, z) of each device onboard the spacecraft is an input, as well as the length of a direct cable linking any two devices. In practice, this physical placement is produced based on other concerns such as thermal constraints or assembly constraints.

In the former approach used prior to this work, the problem was tackled using a two-step procedure, with (1) the dispatch of devices onto matrices, based on a genetic algorithm which tries to group devices which are near from each other in the spacecraft, and (2) the optimization of each order of traversal of devices in each command matrix, based on an home-made Traveling Salesman Problem (TSP) solver using Constraint Programming technology. This paper describes the new techniques developed to solve the problem based on a more global perspective. These new techniques have already been applied to one spacecraft which is currently active and to eight spacecrafts which are currently being built. From an industrial point of view, they led to a 15 % improvement in cable weight, and they allowed to reduce computation times to solve this problem from several days to a few hours.

The rest of the paper is organized as follows: the problem is first related to existing work (Sect. 2), then an integer linear programming formulation is proposed (Sect. 3), approximate search schemes are described (Sect. 4), and finally experimental results on realistic instances are provided (Sect. 5).

2 Problem Analysis

The command architecture design problem considered can be related with the design of electronic devices, which contain several electronic components which must be linked with a controller or with a power source. One key difference here is that the topology considered for spacecrafts is very different from that of electronic devices, as well as the architecture choice.

In another direction, the command architecture design problem can be related to *Vehicle Routing Problems* (VRPs [8]). In a VRP, the inputs are a set of customers placed at some positions and a set of vehicles placed at depots, and the goal is to find vehicle tours which start and end at depots, which visit all customers once, and which minimize the sum of the length of vehicle tours. In our design problem, devices can be seen as customers and each row (resp. each column) of a command matrix can be seen as a vehicle whose tour is the command loop of that row (resp. of that column). For each matrix, there is one depot per row and one depot per column. As these depots may be placed at different locations in the physical architecture of the spacecraft, the problem is

actually a kind of *Multi-Depot Vehicle Routing Problem* (MDVRP). Moreover, as each loop associated with a row (resp. a column) cannot traverse more devices than the number of columns (resp. the number of rows) in a matrix, the problem can be related with the *Capacitated Vehicle Routing Problem* (CVRP), where vehicles have a limited capacity.

However, the design problem to solve has several differences with classical forms of VRPs. The first difference is that some devices (some customers) must be segregated in order to meet reliability constraints. Such kinds of segregation constraints were also considered in [4], where the goal was to compute vehicle routes for vehicles transporting hazardous materials with some constraints on materials which can be put together in a single vehicle. Another difference with standard VRPs is that each device (each customer) needs to be visited twice: once by a command loop associated with a row of a matrix, once by a command loop associated with a column of the same matrix. Moreover, due to the command matrix architecture, devices belonging to the same row cannot belong to the same column, therefore row allocation and column allocation are not independent. Because of this interaction between the VRP on the rows and the VRP on the columns, we call this problem the *Two-dimension Vehicle Routing Problem*. We are not aware of any previous work involving such a two-dimensional aspect, with or without additional segregation constraints and/or heterogeneous vehicle capacities and/or multiple depots. In the following, we propose a formulation for this new problem and we study different resolution techniques.

3 Integer Linear Programming Formulation

To formalize the command loop design problem, we consider the following input data:

- a set $\mathbf{D}$ of devices;
- a set $\mathbf{M}$ of command matrices; each command matrix m contains a set of rows $\mathbf{R_m}$ and a set of columns $\mathbf{C_m}$;
- the set $\mathbf{R}$ of command matrix rows, defined as $R = \cup_{m \in M} R_m$;
- the set $\mathbf{C}$ of command matrix columns, defined as $C = \cup_{m \in M} C_m$;
- the set $\mathbf{P}$ of positions available for placing devices, defined as $P = \cup_{m \in M} (R_m \times C_m)$; in other words, P contains all pairs (r, c) formed by a row and a column belonging to the same matrix; we assume that there are more positions than devices ($|P| \geq |D|$), otherwise the problem is directly inconsistent;
- a set of segregation constraints $\mathbf{Seg} \subseteq D \times D$; a pair (i, j) belongs to Seg when devices i and j must not belong to the same row or to the same column of a command matrix;
- the set of arcs $\mathbf{A^R} = (D \times D) \cup (R \times D) \cup (D \times R)$ containing all possible connections between successive devices on row command loops; for $r \in R$ and $i \in D$, pair (r, i) corresponds to the connection from the controller of the command loop of row r to device i, and pair (i, r) corresponds to the connection from device i to this controller;

- the set of arcs $\mathbf{A^C} = (D \times D) \cup (C \times D) \cup (D \times C)$ containing all possible connections between successive devices on column command loops; for $c \in C$ and $i \in D$, pair (c, i) corresponds to the connection from the controller of the command loop of column c to device i, and pair (i, c) corresponds to the connection from device i to this controller;
- a length function $\mathbf{L}$ such that for any arc $(i, j) \in A^R \cup A^C$, L_{ij} gives the length of a direct cable associated with arc (i, j); lengths specified by L are computed in a preprocessing step using a Floyd-Warshall algorithm on the graph which contains possible cable routes between devices.

Matrix Allocation. In order to model the problem, we first represent the allocation of devices to command matrices. To do this, we consider the following sets of variables:

- $\mathbf{pos_{ip}} \in \{0, 1\}$ $(i \in D, p \in P)$, for representing whether device i is placed in matrix position p (value 1) or not (value 0);
- $\mathbf{row_{ir}} \in \{0, 1\}$ $(i \in D, r \in R)$, for representing whether device i is placed in matrix row r (value 1) or not (value 0);
- $\mathbf{col_{ic}} \in \{0, 1\}$ $(i \in D, c \in C)$, for representing whether device i is placed in matrix column c (value 1) or not (value 0).

Matrix allocation Constraints 1 to 6 are then imposed on these variables. Constraint 1 expresses that a device is allocated to exactly one position in command matrices. Constraint 2 expresses that each matrix position can be associated with at most one device. Constraints 3 and 4 define the row and the column associated with a device from the position at which this device is placed. Last, Constraints 5–6 impose that row and column choices must differ for devices which must be segregated. Note that the capacity constraints on the number of devices involved in a command loop is indirectly expressed by this set of constraints.

$$\forall i \in D, \sum_{p \in P} pos_{ip} = 1 \tag{1}$$

$$\forall p \in P, \sum_{i \in D} pos_{ip} \leq 1 \tag{2}$$

$$\forall i \in D, \forall r \in R, row_{ir} = \sum_{p \in P \,|\, p=(r,c)} pos_{ip} \tag{3}$$

$$\forall i \in D, \forall c \in C, col_{ic} = \sum_{p \in P \,|\, p=(r,c)} pos_{ip} \tag{4}$$

$$\forall r \in R, \forall (i, j) \in Seg, row_{ir} + row_{jr} \leq 1 \tag{5}$$

$$\forall c \in C, \forall (i, j) \in Seg, col_{ic} + col_{jc} \leq 1 \tag{6}$$

Cable Routing. In order to represent how devices are connected to each other, i.e. in order to represent vehicle routing constraints, we introduce two sets of variables:

- $x_{ij}^R \in \{0,1\}$ $((i,j) \in A^R)$, for representing whether arc (i,j) is traversed for connecting i and j on a row command loop;
- $x_{ij}^C \in \{0,1\}$ $((i,j) \in A^C)$, for representing whether arc (i,j) is traversed for connecting i and j on a column command loop.

Constraints 7 to 12 are imposed on these variables. These constraints are standard constraints used for representing vehicle routing problems. Concerning row connections, Constraints 7 and 8 are flow constraints expressing that for each device and each row controller, there is a unique incoming connection and a unique outgoing connection. Constraint 9 corresponds to sub-tour elimination, that is it forbids cable tours which do not contain any command loop controller. These sub-tour elimination constraints can be added incrementally when solving the problem, in order to avoid an exponential blow-up in the problem size. Constraints 10 to 12 impose similar constraints for columns.

$$\forall i \in D \cup R, \sum_{j \mid (i,j) \in A^R} x_{ij}^R = 1 \tag{7}$$

$$\forall i \in D \cup R, \sum_{j \mid (j,i) \in A^R} x_{ji}^R = 1 \tag{8}$$

$$\forall S \subseteq D \text{ s.t. } S \neq \emptyset \sum_{(i,j) \in A^R \mid i,j \in S} x_{ij}^R \leq |S| - 1 \tag{9}$$

$$\forall i \in D \cup C, \sum_{j \mid (i,j) \in A^C} x_{ij}^C = 1 \tag{10}$$

$$\forall i \in D \cup C, \sum_{j \mid (j,i) \in A^C} x_{ji}^C = 1 \tag{11}$$

$$\forall S \subseteq D \text{ s.t. } S \neq \emptyset \sum_{(i,j) \in A^C \mid i,j \in S} x_{ij}^C \leq |S| - 1 \tag{12}$$

Compatibility Between Matrix Allocation and Cable Routing. In order to represent the coupling between matrix allocation and cable routing, we introduce Constraints 13 to 20. Constraints 13 and 14 express that if device j is the successor of device i on the cable route associated with a row, then these two devices must be placed on the same matrix row. Constraint 15 imposes that if there is a connection from the controller of the command loop of row r to device i, then i must be placed on matrix row r. Similarly, Constraint 16 imposes that if device i is connected to the controller of the command loop of row r, then i must be placed on matrix row r. Constraints 17 to 20 impose similar specifications for columns. Note that some of these constraints are redundant.

$$\forall i,j \in D, \forall r \in R,\ row_{ir} + x_{ij}^R \leq row_{jr} + 1 \tag{13}$$

$$\forall i,j \in D, \forall r \in R,\ row_{jr} + x_{ij}^R \leq row_{ir} + 1 \tag{14}$$

$$\forall i \in D, \forall r \in R,\ x_{ri}^R \leq row_{ir} \tag{15}$$

$$\forall i \in D, \forall r \in R,\ x_{ir}^R \leq row_{ir} \tag{16}$$

$$\forall i, j \in D, \forall c \in C,\ col_{ic} + x^C_{ij} \leq col_{jc} + 1 \tag{17}$$

$$\forall i, j \in D, \forall c \in C,\ col_{jc} + x^C_{ij} \leq col_{ic} + 1 \tag{18}$$

$$\forall i \in D, \forall c \in C,\ x^C_{ci} \leq col_{ic} \tag{19}$$

$$\forall i \in D, \forall c \in C,\ x^C_{ic} \leq col_{ic} \tag{20}$$

Global Model. From all previous elements, the global model of the problem corresponds to the following integer linear program, in which the goal is to minimize the total length used for routing cables on row command loops and column command loops:

$$\text{minimize} \sum_{(i,j)\in A^R} x^R_{ij} L_{ij} + \sum_{(i,j)\in A^C} x^C_{ij} L_{ij} \tag{21}$$

subject to :

Matrix allocation constraints (Constraints 1−6)

Vehicle routing constraints (Constraints 7−12)

Allocation/routing compatibility constraints (Constraints 13−20)

$pos_{ip} \in \{0,1\}\ (i \in D, p \in P)$

$row_{ir} \in \{0,1\}\ (i \in D, r \in R), col_{ic} \in \{0,1\}\ (i \in D, c \in C)$

$x^R_{ij} \in \{0,1\}\ ((i,j) \in A^R), x^C_{ij} \in \{0,1\}\ ((i,j) \in A^C)$

Resolution. From the previous Integer Linear Program (ILP), it is possible to use standard solvers such as IBM ILOG CPLEX, by adding sub-tour elimination constraints step by step. As shown in the experiments (see Sect. 5), this approach does not scale well, even when considering only medium size instances. This is why we also defined approximate search schemes (see Sect. 4).

Constraint Programming Formulation. To model this problem, we also tried a pure Constraint Programming (CP) approach. For space limitation reasons, it is presented only at a global level. The CP model built contains:

- for each device $i \in D$, variables $row_i \in R$, $col_i \in C$, $pos_i \in P$, to respectively describe the row, the column, and the matrix position associated with i;
- for each element $i \in D \cup R$, one variable $rnext_i \in D \cup R$ representing the index of the element following i on its row (compared to the ILP model, introduction of integer variables instead of 0/1 variables);
- for each element $i \in D \cup C$, one variable $cnext_i \in D \cup C$ representing the index of the element following i on its column.

Over these variables, several basic constraints are imposed, including:

- *alldifferent* constraints over variables pos_i, to express that two devices cannot be located at the same matrix position;
- *element* constraints linking the row/column of a device with its position;
- constraints expressing that a device cannot follow itself on a row or column;

– row/column segregation constraints for device pairs $(i, j) \in Seg$;
– *element* constraints expressing that the row (resp. column) of a device i is the same as the row (resp. column) associated with $rnext_i$ (resp. $cnext_i$);
– *alldifferent* constraints over variables $rnext_i$ (resp. $cnext_i$).

To improve the power of constraint propagation, as in existing CP models of VRPs/TSPs, the model built also contains redundant variables $rprev_i$ (resp. $cprev_i$) to represent the element which precedes i on its row (resp. on its column), together with alldifferent constraints over these variables and constraints such as $rprev_{rnext_i} = i$ for every $i \in D \cup R$, and $cprev_{cnext_i} = i$ for every $i \in D \cup C$.

Last, the CP model built contains a set of symmetry breaking constraints. First, for every command matrix, it is possible to impose that any row (resp. any column) in this matrix always contains more devices than the next row (resp. the next column) in the matrix. When all matrices share the same dimension, it is also possible to enforce that the number of devices in matrix k cannot be less than the number of devices in matrice $k+1$. Such symmetry breaking constraints were also tested for the ILP model, where they were shown to degrade the results.

Many other CP models could be considered. With the CP model developed, we managed to find solutions, but their quality was not as good as the solutions obtained with ILP, which is why we focus on ILP in the rest of the paper.

4 Two Local Search Approaches

In this section, we present the two local search algorithms we developed and which permit to find good quality solutions within limited computing times. These algorithms start from a complete assignment of the decision variables of the problem and try to iteratively improve the current solution by applying actions towards promising regions of the search space. These actions are local moves that modify the current solution to neighbor solutions. We first present the neighborhood structure which is used, and then the two metaheuristics employed for driving local moves. These two metaheuristics are Simulated Annealing (SA [5]) and Iterated Local Search (ILS [6]).

4.1 Neighborhood Definition

The neighborhood we consider is obtained by combining two kinds of updates:

– *command matrix updates*, which update the allocation of devices to the rows and columns of the command matrices;
– *cable routing updates*, which update the way cables are routed to cover the set of devices belonging to a same row or to a same column.

More precisely, a local move in the neighborhood is performed as follows:

– select one device d placed at a position p, select one position $p' \neq p$ (possibly in a different matrix), and exchange the contents of p and p'; as a result, if position p' selected for reallocating d contains a device d', then after the move,

position p contains device d' and position p' contains device d; if position p' contains no device, then after the move, position p contains no device and position p' contains device d;
- after the previous step, for all rows and columns whose associated set of devices is modified (at most 2 rows and 2 columns concerned), use a TSP routine for improving the routing of cables for these rows and columns.

For the second step, several strategies can be used for updating cable routes. A first possible strategy is to use a complete TSP solver for solving the TSP associated with each impacted row or column. However, this increases the duration required for performing a local move, and such a strategy is effective only for small size problems. A second possible strategy is to use a very fast mechanism which only looks for a good insertion position in cable routes for the device(s) whose rows and columns are updated (computing time linear in the maximum number of rows and columns). However, the cable routes obtained based only on such a greedy insertion rule may lead to a poor quality evaluation of the best cable routes, and therefore to a poor quality evaluation of candidate matrix updates.

To find a compromise between the speed of each move and the quality of the evaluation, we chose, after several experiments, an intermediate strategy which works well on the largest instances (16×16 command matrices). This strategy consists in running the standard *2-opt* heuristic algorithm [3] on the TSP defined by each impacted row and column. Given a TSP tour $[d_1, d_2, \ldots, d_n]$ corresponding to a routing of cables between devices, the idea in 2-opt is to remove at each step two edges (d_i, d_{i+1}) and (d_j, d_{j+1}) from the tour (with $i < j$), and to reconnect the subtours created to consider the new valid tour $[d_1, \ldots, d_i, d_j, d_{j-1}, \ldots, d_{i+1}, d_{j+1}, \ldots, d_n]$ (head and tail of the initial tour kept, and part between d_{i+1} and d_j traversed in the other way around). The move is accepted only if the new tour obtained is shorter, which can be evaluated in constant time. In 2-opt, such moves are applied until no more 2-opt improvement is possible. The advantage of the 2-opt algorithm is that it is known to produce good quality routings within short computing times (complexity quadratic in the number of elements to be covered by the tour). Other approximate TSP resolution schemes could be considered such as *3-opt*, *k-opt* [7] or *or-opt* [1]. The global idea is that faster TSP search permits to do more local moves, while a better quality TSP search permits to choose the most promising local moves.

4.2 Constraint Satisfaction and Criterion Evaluation

In practice, the local search algorithm is implemented by handling integer variables instead of 0/1 variables as in the ILP model. For instance, instead of manipulating variables $pos_{ip} \in \{0, 1\}$ for representing whether device i is placed at position p, we maintain integer variables pos_i representing the position of device i in matrices, as in the Constraint Programming model. Also, for realizing the local search algorithms, we do not consider symmetry breaking constraints because they reduce the accessibility between some neighbor solutions in the search space.

A first key property of the problem considered is that if device segregation constraints are discarded (Constraints 5–6), then it is easy to produce a first solution which satisfies all constraints of the problem. Indeed, it suffices to put device 1 in position 1, device 2 in position 2... and so on until all devices are put in matrices, and then it is possible to run any TSP algorithm on each non empty row and column obtained to get a routing of cables.

A second key property is that the local moves introduced in Sect. 4.1 preserve the satisfaction of all constraints, again except from segregation constraints whose satisfaction can be improved or deteriorated by each local move.

The satisfaction of segregation constraints being non-trivial, we relax the satisfaction of these constraints and manipulate a violation degree instead. To evaluate the overall quality of a solution, this violation degree is combined with the total cable length to get a global score. To do this, we maintain a score for each row r as:

$$scoreRow(r) = cableLengthRow(r) + nSegViolated(r) \times SEG\_COST$$

with $cableLengthRow(r)$ the total length of cables for row r, $nSegViolated(r)$ the number of segregation constraints violated on row r, and SEG_COST a constant factor set big enough for having the satisfaction of segregation constraints preferred to any improvement in cable length at the end of the local search. For each column c, a score $scoreCol(c)$ is defined similarly, and the total score associated with a solution sol is given by:

$$score(sol) = \sum_{r \in R} scoreRow(r) + \sum_{c \in C} scoreCol(c)$$

In other words, compared to the optimization criterion defined in Eq. 21, we add a constraint violated degree to the expression of the score. The score formulation provided also shows that the score can be decomposed by rows and columns, which allows incremental evaluations to be performed when changes occur only on a small part of the problem.

4.3 Simulated Annealing Metaheuristics

To obtain good sequences of local moves with the neighborhood structure defined, it is first possible to consider a standard Simulated Annealing algorithm [5]. The latter corresponds to Algorithm 1. It starts from an initial solution s_0 and at each step, it considers a random solution s' in the neighborhood of the current solution s. Solution s' is accepted as the new current solution if its score is better than the score of s. It is also accepted with some probability when s' deteriorates the score (see lines 6 to 8). Accepting deteriorating moves allows local minima to be escaped. The acceptance probability depends on the criterion deterioration Δ and on the temperature parameter $temp$ of the simulated annealing (use of the Metropolis rule $exp(-\Delta/temp)$). Initially, the temperature is high and many deteriorating moves can be accepted, whereas at the end of the search the temperature is low and almost only improving moves

are accepted. In our case, the temperature is decreased step by step: as shown in lines 4–5, several iterations are performed with the same temperature, and the time spent with a particular temperature depends on the maximum time allowed and on the number of temperature steps required. The solution returned is the best solution found during all iterations.

As usual in local search, setting good values for parameters is not straightforward. In our settings, initial temperature *initTemp* is set using a fast automatic procedure which ensures that at the first temperature step, approximately 80 % of the local moves are accepted. To obtain such a setting, we start from a low temperature and progressively increase this temperature until we estimate that 80 % of the local moves are accepted. To obtain such an estimation, we perform 10000 local moves and we compute the number of these moves which would have been accepted by the simulated annealing acceptance rule. Following experimental evaluations, decreasing factor λ is arbitrarily set to 0.98 and the number of temperature steps *nTempSteps* is set to 1000. Last, the maximum time allocated to the search (*MaxTime*) is left free.

Algorithm 1. SimulatedAnnealing(*initTemp*,λ,*nTempSteps*,*MaxTime*)

Data: *initTemp*: initial temperature, λ: temperature reduction factor, *nTempSteps*: number of temperature reduction steps, *MaxTime*: maximum computing time allowed

1 $tempStep \leftarrow 1$;
2 $s \leftarrow firstSolution()$;
3 **while** $tempStep \leq nTempSteps$ **do**
4 $\quad MaxTimeStep \leftarrow tempStep \cdot (MaxTime/nTempSteps)$;
5 $\quad$ **while** $currentTime() < MaxTimeStep$ **do**
6 $\quad\quad s' \leftarrow selectRandomNeighbor(s)$;
7 $\quad\quad \Delta \leftarrow score(s') - score(s)$;
8 $\quad\quad$ **if** $(\Delta < 0) \vee (rand() < exp(-\Delta/temp))$ **then** $s \leftarrow s'$;
9 $\quad temp \leftarrow \lambda \cdot temp$;
10 $\quad tempStep \leftarrow tempStep + 1$;

4.4 Iterated Local Search Metaheuristics

A second approximate algorithm considered is the Iterated Local Search algorithm (ILS [6]), which has already been applied to Vehicle Routing Problems [4]. This algorithm iteratively tries to find local minima in the search space. More precisely, the ILS algorithm (Algorithm 2) iterates two phases:

- a *local search phase* during which the algorithm searches for a local optimum (lines 3 to 11);
- a *perturbation phase*, during which the local optimum found at the previous phase is perturbed by some random changes (line 12); usually, the perturbation strength must be sufficient for driving the search to another local optimum, and not too large for avoiding performing a kind of random restart.

The solution returned is the best solution found during all iterations.

Local Search Phase Setting. In our settings, the local search phase works on the neighborhood introduced in Sect. 4.1 and tries to select at each step the best neighbor in this neighborhood. As this neighborhood can be quite large (number of neighbors quadratic in the number of matrix positions), we use a neighbor selection strategy which tests only a subset of the candidate local moves. To do this, we first evaluate for each device d a contribution $contrib(d)$ to the global score. This contribution is given by:

$$contrib(d) = contribCableLength(d) + contribSegViolation(d)$$

with $contribCableLength(d)$ the sum of the length of incoming and outgoing cables for device d on its row and its column, and $contribSegViolation(d)$ the number of violated segregation constraints which involve device d times constant factor SEG_COST.

Then, we select a device d which has the highest contribution $contrib(d)$ to the score, and we analyze all neighbors in which d is moved to another position in command matrices (line 7). If the best neighbor found improves the global score, this best neighbor is kept as the new current solution and some device contributions are recomputed before going to the next iteration of the local search. If no best neighbor is found, then the algorithm considers another device d' not considered yet and which has the highest contribution $contrib(d')$. If all devices have already been considered, then a local minimum has been reached and the ILS procedure goes on with the perturbation phase.

Perturbation Phase Setting. For the perturbation phase (line 12), several strategies were tested, like doing an increasing number of random moves or doing many random moves. Experiments showed that for our problem, doing just one random move often suffices to go out from a local optimum. Also, in our problem, making several random moves at each perturbation phase slows down the local search phase because it then takes more time to converge to another local optimum.

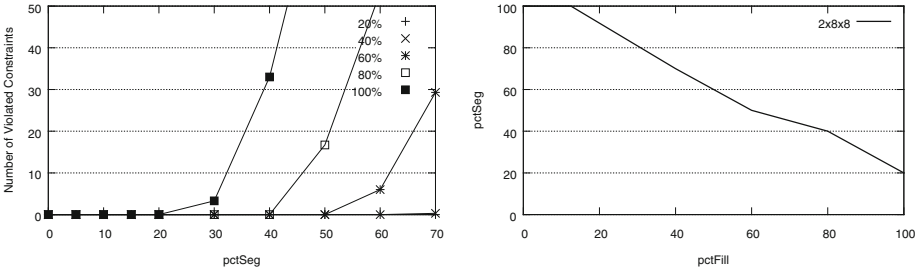
Algorithm 2. IteratedLocalSearch($maxTime$)

```
   Data: MaxTime: maximum computation time allowed
1  s ← firstSolution();
2  while currentTime() < MaxTime do
3      Candidates ← D ;
4      while (Candidates ≠ ∅) ∧ (currentTime() < MaxTime) do
5          contribMax = max_{e' ∈ Candidates} contribs(d');
6          select d in {d' ∈ Candidates | contrib(d') = contribMax};
7          s' ← selectBestNeighbor(s, d) ;
8          if score(s') < score(s) then
9              s ← s';
10             Candidates ← D ;
11         else Remove d from Candidates ;
12     s ← selectRandomNeighbor(s) ;
```

5 Experiments

To compare the different resolution techniques proposed (ILP, SA, ILS), experiments were performed on real instances. We also developed a random instance generator which allowed us to test the different techniques on a wide set of realistic instances. This generator takes several parameters as an input: (1) the number of command matrices **nM** in the instance, (2) the number of rows **nRbM** and the number of columns **nCbM** in each command matrix; (3) the filling percentage of matrices **pctFill**; the number of devices in the instance generated is then automatically computed by $nD = \lfloor pctFill \cdot nM \cdot nRbM \cdot nCbM \rfloor$; distances between devices are generated by computing all pair shortest paths in a graph whose edge weights are given by a uniform random distribution (this way, device distances satisfy the triangle inequality, as in real instances); (4) the percentage **pctSeg** of devices which must be segregated in the instance generated; the number of segregation constraints in Seg is then given by $\lfloor pctSeg \cdot nD\,(nD-1)/2 \rfloor$; devices to be segregated are chosen randomly using a uniform distribution. Figure 2 gives an idea of how the problem becomes more and more constrained when parameters **pctFill** and **pctSeg** are increased. Such results are useful to help space engineers to define the dimensions of the command system.

Experimental Environment. The experimental results presented in the paper only concern randomly generated benchmarks. More precisely, for every fixed parameters settings, we generated at least 5 random instances. As shown in Figures 3, 4 and 5, we also made several experiments to evaluate the sensibility to the variation of some parameters (sensibility to the maximum CPU time allowed in Fig. 5a, and sensibility to the size of matrices in Fig. 5b). In a longer version, we could give more details concerning the influence of parameters settings, and also concerning the influence of algorithmic settings such as the initial temperature of the Simulated Annealing (more exploration of the search space

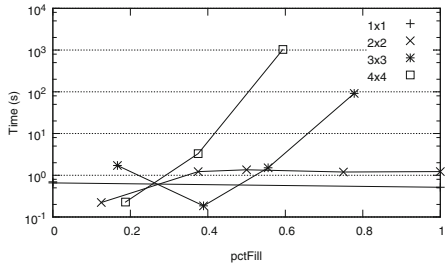


(a) (b)

Fig. 2. (a) Number of violated segregation constraints in the best solution found in one hour, with parameters $nM = 2$ and $nRbM = nCbM = 8$ (the different lines correspond to distinct values of parameter $pctFill$); (b) consistency limit: no consistent solution found for instances corresponding to (pctFill, pctSeg) values located over the line

with a high temperature, faster production of first solutions with a low temperature). Experiments are performed on a processor Intel Xeon CPU W3530 2,80 GHz and 16 GiB of RAM.

Evaluation of the ILP Approach. To evaluate the ILP model presented in Sect. 3, we used IBM ILOG CPLEX 12.5. As shown in Fig. 3(a), based on the ILP model, finding optimal solutions and proving their optimality is only possible for very small instances. For example, when considering only 2 command matrices of size 4×4, there is already an exponential blow-up in computing times when the filling percentage increases. On this point, we believe that the two-dimension aspect of the VRP problems to solve and the segregation constraints make the search space much more combinatorial than in a standard VRP. However, the ILP approach allows us to have some optimum solutions which can be used to evaluate the efficiency of other algorithms on small instances. Such results are shown in Fig. 3(b), where we can see that the solutions obtained using SA and ILS are almost optimal for problems involving only 2×2 matrices which can

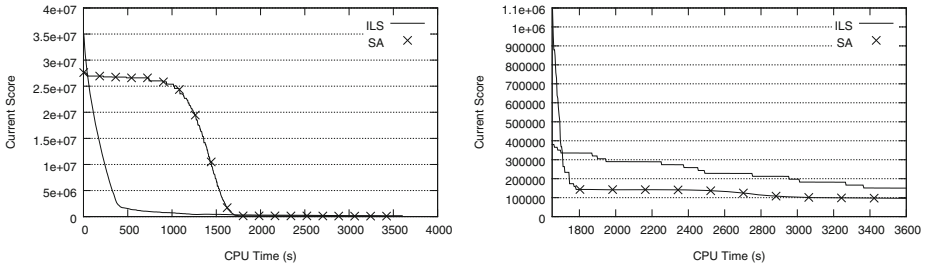


(a)

$2 \times (R_m \times C_m)$	(2×2)	(3×3)	(4×4)
ILP	46612	40020	70402
LB	46612	22719	31184
SA	46916	40560	51907
ILS	46916	40650	52856

(b)

Fig. 3. Some results for fixed parameters $nM = 2$ and $pctSeg = 10\,\%$: (a) computation time required to solve the ILP model proposed for small matrix sizes and for various filling percentages; (b) comparison between ILP (bestScore), ILP LowerBound (LB), SA, and ILS: best score obtained for $pctFill = 80\,\%$ with a maximum computing time $MaxTime = 300s$



(a) (b)

Fig. 4. (a) Evolution of the best known score during search, for parameter setting $nM = 8, nRbM = nCbM = 16, pctFill = 80\,\%, pctSeg = 10\,\%$; (b) zoom on (a)

be solved optimally by ILP. For matrices of size 4×4, Fig. 3(b) shows that the quality of the best solution found by ILP is rather poor compared to the approximate solutions produced by SA and ILS.

Evaluation of SA and ILS. Compared to ILP, the SA and ILS approaches are designed for tackling medium and large instances. Figure 4(a) gives a first comparison between SA and ILS on a large instance involving 8 matrices of size 16×16. This figure shows the evolution of the score associated to the best known solution in function of the computation time. It is possible to see that ILS finds an acceptable solution faster than SA (the steep decreasing of the score corresponds to the satisfaction of all segregation constraints). However, after some computing time, when the temperature of the SA algorithm decreases, SA manages to find an acceptable solution and this solution is better than the solution provided by ILS (see Fig. 4(b)). Such results can be explained by the fact that given the size of the problems considered, the convergence to a local optimum during the local search phase of ILS is rather slow, which entails that the ILS algorithm only visits a small number of local optima. On the opposite, the SA algorithm is looking first for the best area in the solution space without looking for the local optimum at each step, therefore it offers a greater diversity in the traversal of the search space.

Figure 5(a) shows that such conclusions are rather robust to the increase in the allowed computation time, since even for quite high values of the maximum computation time, the SA algorithm still produces better results. For low computation times, it is possible to see that ILS is still working on the satisfaction of the segregation constraints while SA is already improving the total length of cables. Additionally, Fig. 5(b) shows that the dominance of SA over ILS increases with the size of the problem, which again means that with the settings chosen,

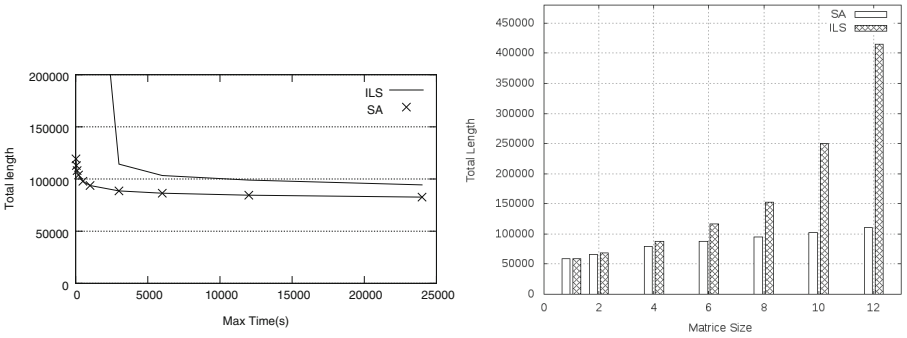


Fig. 5. Comparison between the best solutions produced by SA and ILS: (a) in function of the maximum time allocated to the solver, for fixed parameters $nM = 6, nRbM = nCbM = 16, pctFill = 80\,\%, pctSeg = 10\,\%$; (b) in function of the problem size, for fixed parameters $nRbM = nCbM = 16, pctFill = 80\,\%, pctSeg = 10\,\%$. In all case, the segregation constraints are satisfied.

SA performs a better exploration of the search space which seems to contain many local minima. Another possible explanation to these results is that in the settings chosen, SA adapts its behavior depending on the maximum computation time, thanks to the particular law chosen for decreasing the temperature. On the opposite, ILS does not exploit the maximum computation time information.

6 Conclusion

In this paper, we described the treatment of a design problem from the space domain which corresponds to a kind of Two-dimension Vehicle Routing Problem with segregation constraints. Several algorithms were proposed to solve this problem. Many other algorithmic settings could be considered, even for the Iterated Local Search scheme, but for the moment the best solution found is the Simulated Annealing approach. One of the perspective would be to use CP modeling elements dedicated to TSP problems [2]. A point is that the problem presented in this paper actually makes some simplifications compared to the real problem, which contains other aspects such as decisions on the width of cables used, constrained due to resistance issues. Additional work is required to optimize these other parts of the real problem. However, even with this approximation, it is still interesting to use the work presented in this paper. The Simulated Annealing (SA) is in production and it allowed to save one week in design time, 15 % in cable length, and approximately 10000 euros per satellite.

References

1. Babin, G., Deneault, S., Laporte, G.: Improvements to the or-opt heuristic for the symmetric traveling salesman problem. J. Oper. Res. Soc. **58**, 402–407 (2007)
2. Benchimol, P., van Hoeve, W.-J., Régin, J.-C., Rousseau, L.-M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**, 205–233 (2012)
3. Croes, G.A.: A method for solving traveling salesman problems. Oper. Res. **6**, 791–812 (1958)
4. Hamdi, K., Labadie, N., Yalaoui, A.: An iterated local search for the vehicle routing problem with conflicts. In: 8th International Conference of Modeling and Simulation (MOSIM 2010) (2010)
5. Kirkpatrick, S., Gelatt Jr., C., Vecchi, M.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
6. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics, pp. 363–397. Springer, New York (2010)
7. Rego, C., Gamboa, D., Glover, F., Osterman, C.: Traveling salesman problem heuristics: leading methods, implementations and latest advances. Eur. J. Oper. Res. **211**, 427–441 (2011)
8. Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications, 2nd edn. MOS-SIAM Series on Optimization (2014)