Using VRPSolver to efficiently solve the Differential Harvest Problem

Gabriel Volte, Eric Bourreau, Rodolphe Giroudeau, Olivier Naud

Using VRPSolver to efficiently solve the Differential Harvest Problem

Gabriel Volte[a,*], Eric Bourreau[a], Rodolphe Giroudeau[a], Olivier Naud[b]

[a]*LIRMM, Univ Montpellier, CNRS, 161 rue Ada, Montpellier, 34392, France*
[b]*ITAP, Univ Montpellier, INRAE, Institut Agro, 2 Pl. Pierre Viala, Montpellier, 34000, France*

Abstract

Precision farming in viticulture raises challenging combinatorial issues such as the Differential Harvest Problem. This problem might appear at first as similar to a simple Capacitated Vehicle Routing Problem but it exhibits problem-specific constraints that make this problem much harder to solve and which are discussed in this article. Our objective was to develop efficient exact methods using column generation and VRPSolver™ based models. In order to investigate possibilities of hybridization, two other models were designed: a constraint programming model and a local search model using LocalSolver™. In order to reach good scalability, new valid inequalities and a hybrid solving scheme are proposed. Extensive experiments were performed both on simulated and real data sets. The results are discussed from operational cost point of view and also performance.

Keywords: OR in agriculture, hybrid method, exact methods, branch-cut-and-price

1. Problem description

Motivation of Differential Harvest Problem comes from precision agriculture. The principle of precision agriculture is to take into account the spatial and temporal variability of crop and its environment in order to optimize one or several criteria, such as yield or agricultural input. The information required to perform precision agriculture may be gathered *a priori* and/or in real-time, depending on the application. The Differential Harvest Problem was first introduced by [8], it consists in optimizing the harvest of different grape qualities in a vineyard so that a minimum quantity, denoted by *Rmin*, of good quality grapes is harvested and directed to a specific bin. Maps of grape qualities are known *a priori* in this problem.

*Corresponding author.
Email addresses: gabriel.volte@lirmm.fr (Gabriel Volte), eric.bourreau@lirmm.fr (Eric Bourreau), rgirou@lirmm.fr (Rodolphe Giroudeau), olivier.naud@inrae.fr (Olivier Naud)

Figure 1: Agronomic map for a vineyard with two grapes qualities.

We will give a brief description of the problem. For more details, one may find previous work on the same problem [8, 20, 24].

A vineyard can be mapped (see Figure 1) by distinguishing areas according to the quality of the grapes. In the vineyard, two grape qualities can be distinguished: $\mathcal{A}$-grapes and $\mathcal{B}$-grapes. Without loss of generality, we consider that $\mathcal{A}$-grapes correspond to the higher grapes quality and $\mathcal{B}$-grapes to the lesser quality. The zones of a row can be represented by an alphabet $\Sigma = \{\mathcal{A}, \mathcal{B}\}$. As it can be observed in Figure 1, a row is a succession of $\mathcal{AB}$ or $\mathcal{BA}$ transitions (two successive $\mathcal{A}$ or $\mathcal{B}$ zones are merged). Therefore, four possible configurations for the representation of each row can be considered:

- $(\mathcal{AB})^p$ (resp. $(\mathcal{BA})^p$), a succession of p $\mathcal{AB}$-transitions (resp. $\mathcal{BA}$-transitions).
- $(\mathcal{AB})^p\mathcal{A}$ (resp. $(\mathcal{BA})^p\mathcal{B}$), a succession of p $\mathcal{AB}$-transitions (resp. $\mathcal{BA}$-transitions) and one $\mathcal{A}$-zone (resp. $\mathcal{B}$-zone) at the end.

Geolocated harvesting machines fitted out with two hoppers (harvest tanks with a capacity of *Cmax*) and mechanisms to direct the harvest in a chosen hopper can handle the vineyard map. In order to differentiate both hoppers, we call one $\mathcal{A}$-hopper and $\mathcal{B}$-hopper the other.

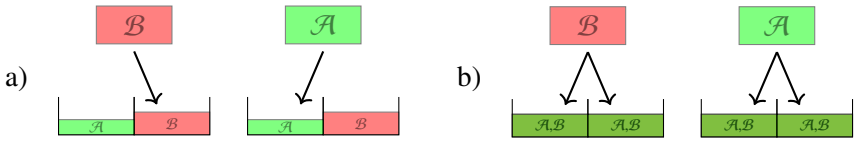


Figure 2: Illustration of the grape harvesting in the hoppers depending on harvesting mode, a) selective mode, b) non-selective mode.

For sake of simplicity, let us define two harvesting modes for these machines, modes that can only be changed when emptying hoppers at the bin:

Definition 1.

- *The selective mode*, cf. Figure 2 a), indicates that grapes are sorted in hoppers accordingly to their quality: $\mathcal{A}$-hopper should receive only $\mathcal{A}$-grapes; $\mathcal{B}$-hopper should receive only $\mathcal{B}$-grapes and a minimal quantity of $\mathcal{A}$-grapes degraded to $\mathcal{B}$-grapes due to a technicality defined below (latency).

- The *non-selective mode*, cf. Figure 2 b), refers to the harvesting mode where grapes are mixed in both hoppers so that the load of each hopper is balanced (equivalent to only one hopper with doubled capacity).

When one of both hoppers is full, both must be emptied into a bin located at one edge of the plot. The selective harvesting mode is needed to harvest separately the *Rmin* quantity of $\mathcal{A}$-grapes, afterwards the machine may change to the non-selective mode where the loading is ideally handled (as one hopper with $2Cmax$ capacity cf. Figure 2).

The objective is to harvest the entire field while minimizing the total harvesting time, which is composed of the harvesting time in the rows and the machine travel time outside rows (between rows or to the depot). Because harvester speed is constant and fixed when harvesting, harvesting time is a fixed cost. Thus, what is to be optimized is the total travel time of the harvester.

Please note that the *selective mode* may be defined slightly differently in related works cited above to handle hopper capacity more flexibly while prioritizing the sorting of $\mathcal{A}$-grapes. This is discussed in section 2 with the overflow management notion.

Let us consider n the rows number in the vineyard and $N = \{(2i-1, 2i) | i \in 1, \ldots, n\}$ the set of rows matched by their extremities. The Differential Harvest Problem may be seen as a Heterogeneous Fleet Vehicle Routing Problem variant (see [1] for a general variant). The vehicle types set is defined by $K = \{s, \overline{s}\}$, with s corresponding to the selective mode and $\overline{s}$ to the non-selective mode and Q^k be the capacity of vehicle type $k \in K$ with $Q^s = Cmax$ and $Q^{\overline{s}} = 2Cmax$.

Let us introduce $\mathcal{R} = \{\mathcal{A}^s, \mathcal{B}^s\} \cup \{(\mathcal{AB})^{\overline{s}}\}$ the set of resources.

- $\mathcal{A}^s$ (resp $\mathcal{B}^s$) the resource representing the $\mathcal{A}$-grapes (resp. $\mathcal{B}$) harvested by vehicles of type s,

- $(\mathcal{AB})^{\overline{s}}$ the resource representing the $\mathcal{A}$-grapes and $\mathcal{B}$-grapes harvested by vehicles of type $\overline{s}$,

As will be seen in the next section, the harvested quantity can be different according to the harvesting direction.

1.1. Latency

There is a technical issue that leads to an asymmetrical harvest depending on the harvesting direction of the row. Before explaining this technical issue, some understanding on how the harvesting machine gathers grapes inside a row is needed. Most grapes next to the machine are shaken by rods in the picking head, picked and quickly directed

to their way towards the harvesting hopper. Yet, some grapes can be stuck in their conveyance to the harvesting hopper due to the picking head size and transfer time of grapes within the mechanisms associated to the picking head.

In this case, the technical issue called *the latency* happens. This problem occurs, only in selective harvesting mode, when changing zones especially when transitioning from a $\mathcal{B}$-zone to a $\mathcal{A}$-zone. We suppose that it is bound with the harvesting machine size. Recall that, in selective mode, $\mathcal{A}$-grapes cannot be corrupted with $\mathcal{B}$-grapes.

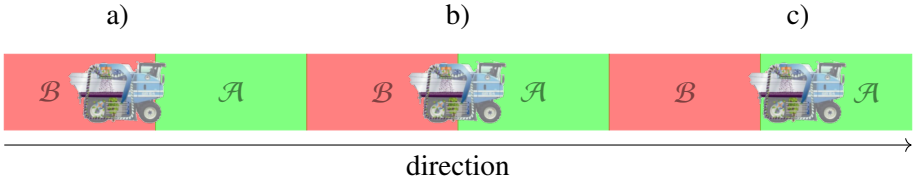


Figure 3: Illustration of the latency technical issue.

Let us observe how the harvesting hopper changes are made according to zone transitions. During a $\mathcal{BA}$-transition, three cases can be considered (cf. Figure 3):

- Cases a) and b), the head of the machine just entered the $\mathcal{A}$-zone or the harvesting machine is overlapping both zones. The harvesting hopper cannot be changed yet (it is still the $\mathcal{B}$-hopper) since $\mathcal{B}$-grapes are still on their way to the hopper due to the above hypothesis.

- Case c), the harvesting machine is completely in the $\mathcal{A}$-zone, no $\mathcal{B}$-grapes are still carried into the harvesting machine. Therefore, the harvesting hopper can be changed (from $\mathcal{B}$-hopper to $\mathcal{A}$-hopper) without corrupting any $\mathcal{A}$-grapes.

During a $\mathcal{AB}$-transition, to avoid corrupting $\mathcal{A}$-grapes with $\mathcal{B}$-grapes, it is enough to change the harvesting hopper as soon as the machine enters into the $\mathcal{B}$-zone. In this case, according to harvesting hypothesis, the quantity of $\mathcal{A}$-grapes leftover inside the machine is considered as negligible, and we assume that no $\mathcal{A}$-grape is lost.

Definition 2. The *latency* corresponds to the delay before changing the harvesting hopper during a $\mathcal{BA}$-transition, in selective mode, avoiding $\mathcal{A}$-grapes corruption with the $\mathcal{B}$-grapes that have not already been carried into the $\mathcal{B}$-hopper.

We show below that some row compositions are symmetrical with regards to harvested quantities of $\mathcal{A}$-grapes and $\mathcal{B}$-grapes in selective mode, and others asymmetrical. For asymmetrical cases, we show that the difference between harvested quantity of grapes of each type according to the harvesting direction is bounded.

Let us note l_i^d, the number of $\mathcal{BA}$-transitions in the row i according to the direction $d \in \{0, 1\}$, with 0 representing the left to right direction and 1 the right to left. Let δ be the $\mathcal{A}$-grapes quantity corrupted into $\mathcal{B}$-grapes during a $\mathcal{BA}$-transition.

Lemma 1. *For each row i, the difference of $\mathcal{BA}$-transitions between both direction is at most one:* $0 \leq |l_i^0 - l_i^1| \leq 1$.

Proof. Four row configurations can be examined:

- In the row configuration $i = (\mathcal{AB})^p, p \geq 1$, if the machine follows the $\mathcal{AB}$-transition (direction 0), then row i can be rewritten as $i = \mathcal{A}(\mathcal{BA})^{p-1}\mathcal{B}$, thus $l_i^0 = p - 1$. In the other direction, the row configuration is $i = (\mathcal{BA})^p$ therefore $l_i^1 = p$ and the difference is $|(p-1) - p| = 1$.
- In configuration $i = (\mathcal{AB})^p\mathcal{A})$ and $(\mathcal{BA})^p\mathcal{B}$, the row can be rewritten as $i = \mathcal{A}(\mathcal{BA})^{p-1}\mathcal{BA}$ and $i = (\mathcal{BA})^p\mathcal{B}$. Thus for both configurations $l_i^0 = p$. In the opposite direction, the number of $\mathcal{BA}$-transitions is the same ($l_i^1 = p$) because rows are symmetrical, and the difference is $|p - p| = 0$.

□

The difference between the harvested quantities of $\mathcal{A}$-grapes and $\mathcal{B}$-grapes is proportional to the number of $\mathcal{BA}$-transitions in a row. However, for a given asymmetrical row, the difference between the harvested quantities of grapes of each type according to the direction of harvest is δ. And the difference is null for symmetrical rows. Thus, only the harvesting direction of the asymmetric rows influences the *Rmin* constraint.

1.2. A specific constraint: the alternate harvesting

In this section, we discuss a specific constraint, hereafter called *alternate harvesting*, imposed in [8] and later in [19] stating that when the machine harvests a row in a direction, it must traverse the next row in the opposite direction.

Definition 3. The *alternate harvesting* (AH) is a constraint stating that when harvesting a row in a given direction, next row has to harvested in the opposite direction.

The idea of AH is to reduce the travel time of the harvester that seems unproductive (when not harvesting grapes). The term *non-alternate harvesting* (NAH) refers to the harvesting of vineyard without forcing the change of direction at each row.

In order to differentiate the movements of the machine between the ends located on the same side of the field and those between opposite ends, we define the concept of *diagonal edges/arcs*.

Definition 4. The set of *diagonal edges/arcs*, denoted $\mathcal{D}$, is a set of edges/arcs corresponding to the paths of the machine crossing the field without harvesting a row.

For the case where *diagonal edges* are forbidden (AH), a logistical issue arises: what happens when hoppers are not full, but it is not possible to find a row that can be harvested in the expected direction? The answer is to return to the depot to empty hoppers and start in the appropriate direction, which incurs additional traveling costs.

It may seem counter-intuitive to cross the field without harvesting the grapes in a row. Yet, allowing these edges brings flexibility to the routes from an operational point of view. For this reason, and unless otherwise stated, in the rest of the article, we consider the harvest as non-alternate.

2. Related Works

In the last years, many works were performed in digital and precision agriculture using operation research techniques to solve vehicle routing problems, scheduling problems, or stochastic problems.

The surveys [3], and [4] assert that most of the agricultural applications involve the motion of machines; hence they first classify the agricultural field operations, which can be modelled as a vehicle routing problem. The authors present an approach to represent the planning and scheduling of moving machines as a vehicle routing problem with time windows where the machine needs to process deterministic, stochastic or dynamic requests.

Vehicle routing problem specific optimization methods are addressed in [21] to solve their fleet routing problem; they have reduced the operating time by at most 17.3% using their approach rather than a human-made solution.

One can find some works on field coverage problem [10, 17], where the goal is to cover a field under technical constraints. The purpose of the work presented in [17] is to reduce soil compaction. Smaller vehicles are used to cover the fields, but those vehicles have smaller storage capacity. Thus the full field coverage in a single run is no longer possible, and a path planning strategy is used to partially cover the field under compacted area minimization constraints. In comparison, the article [10] focuses both on finding minimal operating cost and on balancing the vehicle's workload.

A column generation approach is used to solve a crop rotation scheduling problem to produce a pre-determined demand for crops while respecting some ecological production constraints; this problem is called the sustainable vegetable crop demand-supply problem [13].

The Differential Harvest Problem has a 2 Dimensional Vector Packing Problem (2-DVPP) [14] component, and a routing component.

Instead of a classical column generation decomposition scheme of a cVRP in which the pricing problem has a knapsack component (with the same routing component), the Differential Harvest Problem has a 2-DVPP pricing problem because selective routes need to handle both grape qualities separately. In [22], the authors found some lower bounds for 2-DVPP and performed numerical experiments using a branch-and-bound procedure. In [11], the authors introduced new lower bounds and compared heuristics and exact methods to solve instances with 25, 50, 100 and 200 items with weights randomly generated among 10 classes. For some classes, even instances with 50 items were not solved exactly. The authors of [9] used a consistent neighborhood search for the bin

packing problem and also for the 2-DVPP on the same benchmark presented above, they always obtained optimal or best known solutions.

The Differential Harvest Problem has been solved with methods based on constraint programming (CP) in a few available papers. First, a step model was proposed in [7] solving, optimally, a 16 rows instance in 6 days. Later a precedence model was introduced in [8], which provided faster results. Still, instances with 14 rows could not be solved within a 2 hours time limit. Other authors proposed to use the Cost-Optimal Reachability Analysis [19] (CORA, which is a branch of model-checking techniques). This technique allowed to find an optimal solution on vineyards with up to 12 rows in few minutes. These authors later enhanced their method, allowing them to find a optimal solution for instances with up to 14 rows [20] with a limited usage of memory.

All above cited papers considered the problem variant in which (i) the harvesting machine has to change direction after harvesting any row (AH) and in which (ii) some $\mathcal{A}$-grapes can be spoiled in the $\mathcal{B}$-hopper while in selective harvesting mode. The latter point, a variation on the definition of selective mode, may be expected on the first hand as a pertinent relaxation of the Differential Harvest Problem problem. As mentioned in [8], when the $\mathcal{A}$-hopper is full, it would be possible to transfer the excess $\mathcal{A}$-grapes into the $\mathcal{B}$-hopper if and only if the *Rmin* constraint can still be satisfied. We call this harvest management option *overflow management*.

The main idea of overflow management is to avoid, as much as possible, emptying hoppers when the $\mathcal{B}$-hopper is not full. Yet, in practice, lesser quality grapes ($\mathcal{B}$-grapes) represent more yield in the field than $\mathcal{A}$-grapes, and we observed on the studied instances that it is the capacity of the $\mathcal{B}$-hopper that is limiting the efficiency of harvest, not the capacity of the $\mathcal{A}$-hopper. We thus investigate the variant of the selective mode where the $\mathcal{A}$-grapes and $\mathcal{B}$-grapes are both strictly sorted in distinct hoppers.

3. Models description

We provide in this section the graph representation that we chose for most of our approaches. We will then introduce the models implemented to solve the Differential Harvest Problem, starting with (1) a compact formulation based on heterogeneous fleet routing problem, followed by (2) a branch-cut-and-price approach using VRPSolver library [15], (3) a local search approach using LocalSolver library [2] and (4) a constraint programming approach introduced in [24].

All the mentioned models, which are derived from several paradigms, are strictly equivalent with regards to the problem solved. They will be compared to each other for performance, and hybridized, in section 5.

The key point in our graph representation is to remove arcs representing rows without losing any information. Let us consider the graph $G = (V, A)$ with $V = \{0, 1, \ldots, 2n\}$ the set of vertices (the row extremities), 0 representing the depot, and A the set of arcs. A harvesting direction for the row can be associated to each row extremity. By convention

and without loss of generality, for each row $(i, j) \in N$, let us say that vertex i (resp. j) corresponds to the harvesting direction 0 (resp. 1). The consumption of resource $r \in \mathcal{R}$ on the vertices, $q_i(r)$ and $q_j(r)$ for all rows $(i, j) \in N$ with regards to the harvesting direction can now be defined. This graph is almost complete; only arcs representing rows are not included in A. The goal is to find a set of routes covering all the rows (see Figure 4) and minimizing the total harvesting cost.

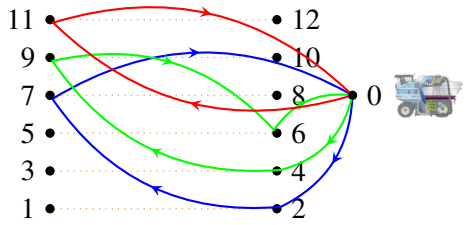


Figure 4: Illustration for a 3 routes, 6 row solution on a graph with 13 vertices and 72 arcs. Dotted edges correspond to rows in the physical representation of the field.

3.1. A heterogeneous fleet vehicle routing problem flow based on a compact formulation

We adapt here the compact formulation described in [5] for the capacitated vehicle routing problem in order to fit the Differential Harvest Problem.

The flow decision variables become $x_a^k,\ k \in K = \{s, \overline{s}\},\ a \in A$, with $x_a^k = 1$ if and only if a vehicle of type k takes arc a.

By a liberal use of notations, each resource is associated here with the vehicle type that harvests it (i.e. $\mathcal{A}^s$, $\mathcal{B}^s$ to vehicle type s and $(AB)^{\overline{s}}$ to $\overline{s}$).

The variables $y_i^r,\ i \in V,\ r \in \mathcal{R}$ represents the accumulated consumption of resource r after visiting vertex i following the path from the depot.

Model 1: Compact Formulation

$$\text{Min} \sum_{k\in K}\sum_{a\in A} x_a^k c_a \tag{3.1}$$

$$\sum_{k\in K}\sum_{a\in\delta^-(i,j)} x_a^k = 1 \qquad \forall (i,j)\in N \tag{3.2}$$

$$\sum_{a\in\delta^-(h)} x_a^k - \sum_{a\in\delta^+(h)} x_a^k = 0 \qquad \forall h\in V, \forall k\in K \tag{3.3}$$

$$l_k \le \sum_{a\in\delta^+(0)} x_a^k \le u_k \qquad \forall k\in K \tag{3.4}$$

$$y_i^r + q_i(r) * x_{i,j}^r - Q^r(1 - x_{i,j}^r) \le y_j^r \qquad \forall (i,j)\in A,\ r\in\mathcal{R} \tag{3.5}$$

$$\sum_{i\in V}\sum_{a\in\delta^+(i)} x_a^s q_i(\mathcal{A}^s) \ge Rmin \tag{3.6}$$

$$x_a^k \in \{0,1\} \qquad \forall a\in A, \forall k\in K \tag{3.7}$$

$$q_i(r) \le y_i^r \le Q^r \qquad \forall i\in V,\ r\in\mathcal{R} \tag{3.8}$$

The function $\delta^-(i,j) = \delta^-(i)\cup\delta^-(j)$ gives the set of arcs entering i or j. The objective function (3.1) minimizes the total harvesting cost of the vineyard. The constraints (3.2) indicate that each row appears only once in a solution, and thus with only one harvesting direction. According to the graph, this means that only one extremity of the row belongs to any feasible solution. Equations (3.3) give the flow conservation constraints. They verify that once a vehicle enters a row, it must leave it. The constraints (3.4) give the lower and upper bounds on the number of vehicle in any solution for each vehicle type.

The constraints (3.5) ensure that routes of any type are not segmented and that the total vehicle load does not exceed the vehicle capacity. These constraints verify that for any resource r, the vehicle load for r is increasing along the path from the depot, i.e. if a vehicle uses the arc (i,j) then the vehicle load after vertex j has to be greater or equal than the load after i, $y_i^r \le y_j^r$. Moreover, with constraints (3.8), y_i^r variables are bounded by the capacity of the vehicle of type k^r.

The constraint (3.6) guarantees that the *Rmin* quantity of $\mathcal{A}$-grapes has been harvested by the selective routes. The flow variables are binary variables (3.7) and the resource consumption variables (3.8) are continuous variables.

3.2. A VRPSolver based model

VRPSolver [15] is a generic branch-cut-and-price solver for vehicle routing problems offering a range of available optimization techniques in the literature (developed specifically for certain problem variants). The operational objective of VRPSolver is to provide a generic framework to efficiently solve vehicle routing problems that allow obtaining solutions of equivalent quality to the best state-of-the-art solutions.

The main idea of VRPSolver is to use a generic structure to model vehicle routing problems: the Packing Set. A Packing Set corresponds to a subset of arcs (resp. vertices) such that in at least one optimal solution, at most, one arc (resp. vertex) of each subset belongs to the solution. For more details on the Packing Set, see Section 3 of [15].

As shown in [12], the path enumeration technique is important in solving vehicle routing problems, especially for small and medium instances or instances with good linear relaxation. To take full advantage of VRPSolver, we need to enable the path enumeration. However, with the graph presented in this section so far, it is not possible to use enumeration because of the sufficient condition that is required for solving with path enumeration [15] (see section 3.4 Path Enumeration).

Indeed, let us recall that the sufficient condition for using path enumeration is:

Every two feasible partial elementary paths (according to Packing Set*) starting in the source that end in the same vertex and map to different coefficients in some essential constraints (see constraints (2b) in [15] p.9) should have visited different subsets of the* Packing Set.

Briefly and informally, essential constraints are the ones which structure the solutions of the problem to be solved at master problem level. In a Heterogeneous Fleet Vehicle Routing Problem, these are related to the effective covering of clients and associated services (here to harvest all rows and sort *Rmin* $\mathcal{A}$-grapes). In our VRPSolver based model for Differential Harvest Problem, model 2, and because the vehicle type (selective or not) can be switched on the harvester anytime it reaches the depot, the essential constraints are given in equations 3.10 and 3.11.

In order to get a Packing Set that satisfies the sufficient condition for enabling enumeration, we chose to add vertices in the graph and to modify the Packing Set so that routes using different subsets of the Packing Set harvest rows in different directions.

Construction 1. *Let us transform* $G = (V, A)$ *in a new graph* $G^* = (V^*, A^*)$*. First, let us duplicate all vertices* i *except for the depot into a new one* i^**. Then, let us add an arc from* i *to* i^* *and transfer all outgoing arcs of* i *to* i^**.*

The packing set can now be updated to take into account the vertex duplication:

$$\mathcal{P}^{\mathcal{V}} = \cup_{(i,j)\in N}\{\{i_k, j_k : k \in K\}, \{i_k^* : k \in K\}, \{j_k^* : k \in K\}\}$$

Vertices representing rows remain in same packing set and each duplicated vertex belongs to a new packing set in order to differentiate the harvesting direction.

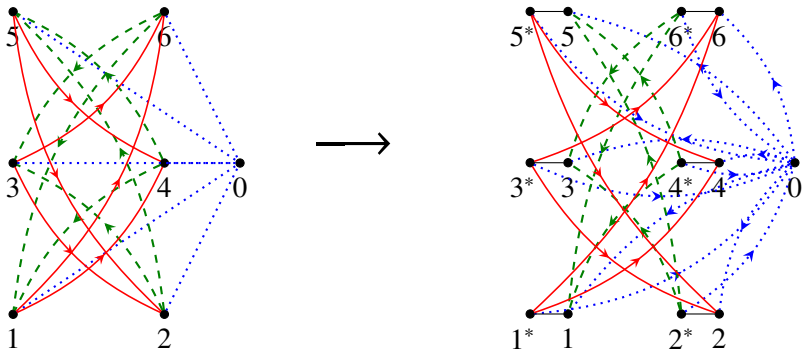


Figure 5: Illustration of graph transformation given by Construction 1.

We now give the proof that the previously defined packing set satisfies the sufficient condition:

Lemma 2. *Using graph $G^* = (V^*, A^*)$, the sufficient condition in [15] Section 3.4 is verified.*

Proof. The proof proposed here is by contradiction. Suppose that there are two partial elementary paths r_1 and r_2 starting at the source, ending at the same vertex and using the same *packing set* but having different coefficients in some essential constraints (i.e. in the Differential Harvest Problem the *Rmin* constraint, equation 3.11) of the master problem. Since that these routes pass through the same *packing set*, coefficients in the covering constraint are identical. Thus, the only constraint that could cause the condition to be invalid is the *Rmin* constraint. The only arc going out of any vertex i is its duplicate vertex i^*, so each route is composed of a succession of vertex pairs (i, i^*). As the set of visited packing sets is identical for r_1 and r_2, all pairs of vertices that compose the route r_1 belong to the route r_2 and vice versa. It implies that all rows have been crossed in the same direction, but in a different order because the source vertices and the sink vertices belong to different *packing set*. Furthermore, coefficients in constraint *Rmin* are identical, which gives us a contradiction. □

Let us now introduce the set of graphs $\{G^k = (V^k, A^k) | k \in K\}$, that will be used by VRPSolver to generate routes.

For each graph G^k, the source and the sink of any route $v^k_{source} = v^k_{sink} = 0$ are defined by the same vertex corresponding to the depot.

The resource consumption q on each arc $a = (i, j) \in A$ is given by $q(a, \mathcal{A}^s) = q_i(\mathcal{A})$, $q(a, \mathcal{B}^s) = q_i(\mathcal{B})$ for selective vehicles and $q(a, (\mathcal{AB})^{\bar{s}}) = q_i(\mathcal{A}) + q_i(\mathcal{B})$ for non-selective routes.

The accumulated resource consumption interval on vertex $i \in V^k$ is $[l_{i,\mathcal{A}^s}, u_{i,\mathcal{A}^s}] = [l_{i,\mathcal{B}^s}, u_{i,\mathcal{B}^s}] = [0, Cmax]$ for both resources for selective routes and $[l_{i,(\mathcal{AB})^{\bar{s}}}, u_{i,(\mathcal{AB})^{\bar{s}}}] = [0, 2 * Cmax]$ for non-selective routes.

The minimum and maximum numbers of selective vehicles is given by $L^s = \left\lceil \frac{Rmin}{Cmax} \right\rceil$, $U^s = n$ and by $L^{\bar{s}} = 0$, $U^{\bar{s}} = n$ for the non-selective vehicles.

The binary variables $x_a^k,\ a \in A^k,\ k \in K$ indicates if arc a is used in the graph G^k.

Similarly to [16], we give in model 2 the following complete formulation used to model DHP with VRPSolver. First, the definition for graphs G^k are recalled, then the ILP formulation of the Restricted Master Problem is given as well as the mapping (the M function) link between arcs $a \in A^k$ and binary variables x_a^k:

Model 2: VRPSolver model for Differential Harvest Problem

Graphs $G^k = (V^k, A^k) = G'$; $v^k_{source} = 0$; $v^k_{sink} = 0,\ k \in K$; $\mathcal{R} = \{\mathcal{A}^s, \mathcal{B}^s, \mathcal{AB}^{\bar{s}}\}$; $q(a, \mathcal{A}^s) = q_i(\mathcal{A})$, $q(a, \mathcal{B}^s) = q_i(\mathcal{B})$, $q(a, (\mathcal{AB})^{\bar{s}}) = q_i(\mathcal{A}) + q_i(\mathcal{B}),\ a = (i, j) \in A^k,\ k \in K$; $[l_{i,\mathcal{A}^s}, u_{i,\mathcal{A}^s}] = [l_{i,\mathcal{B}^s}, u_{i,\mathcal{B}^s}] = [0, Q^s]$, $[l_{i,(\mathcal{AB})^{\bar{s}}}, u_{i,(\mathcal{AB})^{\bar{s}}}] = [0, Q^{\bar{s}}]$, $i \in V^k,\ k \in K$;
Binary variables $x_a^k,\ a \in A^k,\ k \in K$. The formulation is :

$$Min \quad \sum_{k \in K} \sum_{a \in A^k} c_a^k x_a^k \tag{3.9}$$

$$S.t \quad \sum_{k \in K} \sum_{a \in \delta^-(i,j)} x_a^k = 1 \qquad \forall (i, j) \in N \tag{3.10}$$

$$\sum_{(i,j) \in A^s} q_i(\mathcal{A}^s) x^s_{(i,j)} \geq Rmin \tag{3.11}$$

$$x_a^k \in \{0, 1\} \qquad \forall k \in K, a \in A^k \tag{3.12}$$

$M(x_a^k) = \{a\},\ a \in A^k,\ k \in K$; $L^s = \left\lceil \frac{Rmin}{Cmax} \right\rceil$, $L^{\bar{s}} = 0$, $U^s = U^{\bar{s}} = n$.

$$\mathcal{P}^{\mathcal{V}} = \cup_{(i,j) \in N} \{\{i_k, j_k : k \in K\}, \{i_k^* : k \in K\}, \{j_k^* : k \in K\}\}$$

Branching is done first on the vehicle number of each type, then on the assignment of a row to a vehicle type and finally on the **x** variables corresponding to the arcs of the graphs. Enumeration is on.

The objective function (3.9) is still to minimize the total cost of the routes. Thus the constraint (3.10) verifies that each row is harvested exactly once. The constraint (3.11) checks that the *Rmin* threshold of $\mathcal{A}$-grapes is harvested with the selective routes.

3.3. The LocalSolver model

LocalSolver [2] is a generic optimization solver, combining exact and heuristics techniques, providing efficient and scalable modelling tools. The key point of LocalSolver is to obtain a good quality solution in a reasonable time compared to mixed-integer linear programming, constraint programming or nonlinear programming solvers.

The same graph $G = (V, A)$ as for the compact formulation (see Section 3.1) will be used for this model. Let us define set variables v_l, each set containing the list of rows belonging to route $l \in \{1, \ldots, n\}$ and boolean variables $s_l = 1$ if and only if the route l is selective and 0 otherwise. The set variables v_l can be seen as binary value vectors, (a_i^l) indicating if an extremity i, $\forall i \in v_l$, belongs to this route or another. The cost of the route l is given by c_l.

Model 3: LocalSolver model

$$\min \sum_{l=1}^{n} c_l \tag{3.13}$$

$$\sum_{l=1}^{n} (a_i^l + a_j^l) = 1 \qquad \forall (i,j) \in N \tag{3.14}$$

$$\text{disjoint}(v_1, \ldots, v_n) \tag{3.15}$$

$$\sum_{i=1}^{2n} q_i(\mathcal{A}^s) * a_i^l * s_l \leq Cmax \qquad \forall l \in \{1, \ldots, n\} \tag{3.16}$$

$$\sum_{i=1}^{2n} q_i(\mathcal{B}^s) * a_i^l * s_l \leq Cmax \qquad \forall l \in \{1, \ldots, n\} \tag{3.17}$$

$$\sum_{i=1}^{2n} (q_i((\mathcal{AB})^{\bar{s}}) * a_i^l \leq 2 * Cmax \qquad \forall l \in \{1, \ldots, n\} \tag{3.18}$$

$$\sum_{l=1}^{n} \sum_{i=1}^{2n} q_i(\mathcal{A}^s) * a_i^l * s_l \geq Rmin \tag{3.19}$$

The objective function (3.13) minimizes the total cost of routes. The disjunctive constraints (3.14) ensure that each row is harvested only once. The capacity constraints (3.16), (3.17) and (3.18) guarantee that vehicle capacity is respected. The constraint

(3.19) verifies that the *Rmin* quantity of $\mathcal{A}$-grapes has been harvested with the selective vehicles. The redundant constraint (3.15) ensures that routes are disjointed and helps the solver obtaining better solutions faster.

3.4. A constraint programming model

Our constraint programming approach is based on the successor/rank/assignation variables model as given in [6]. In our model tailored to DHP are included variables measuring the cumulative sum of harvested grapes of each type, in order to handle capacity constraints. In a previous work [24], we have proposed a constructive search strategy that mimics a nearest neighbor heuristic. The purpose of this strategy is to find a good solution in a few decision nodes. One can find more details on the model and the search strategy in [24].

4. Bounds on diagonal arcs

A lower bound on the number of rows that have together specific properties, subsequently called *limiting rows*, is introduced in this section. In a feasible solution, these rows must be harvested according to a common direction and can't be associated in routes of a solution with rows harvested in the other direction. Thus, these rows require the use of diagonal arcs in the solution. This bound will add, in our optimization models, information on direction of a subset of rows imposed when the value of *Rmin* is high. The high influence of this bound on experimental results will be exhibited in Section 5.2.3.

Let us define the notion of *round trip* and row harvested in the *good direction* (resp. *wrong direction*).

Definition 5. A *round trip* is a cycle starting from the depot harvesting two rows without diagonal arcs.

Definition 6. A row $i \in N$ is harvested in the *good direction*, denoted τ, (resp. *wrong direction*, denoted $\overline{\tau}$), if it is traversed in direction which maximizes (resp. minimizes) the $\mathcal{A}$-grapes harvested.

To differentiate the ends of the rows, we define $N^l = \{(i, j) \in N,\ q_i(\mathcal{A}) > q_j(\mathcal{A})\}$ as the rows to be harvested in the good direction from left to right, and $N^r = \{(i, j) \in N,\ q_i(\mathcal{A}) < q_j(\mathcal{A})\}$, as the rows to be harvested in the good direction from right to left. The set $N^= = \{(i, j) \in N,\ q_i(\mathcal{A}) = q_j(\mathcal{A})\}$ denotes the set of symmetric rows with $n^= = |N^=|$. We denote $n^l = |N^l|$, $n^r = |N^r|$ and we have $n = n^l + n^r + n^=$.

For example, in Figure 7 (with the physical representation) the cycle $\{0, 2, 1, 3, 4, 0\}$ shapes a *round trip* but not path $\{0, 2, 1, 4, 3, 0\}$ because $(1, 4)$ is a diagonal arc.

Rows belonging to $N^=$ have the same amount of grapes in both directions, so these rows are always harvested in the good direction.

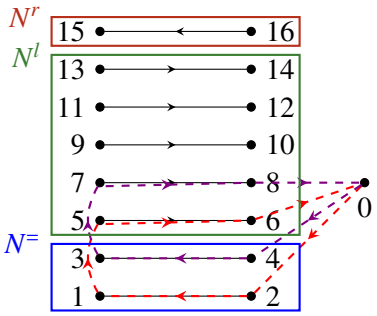


Figure 6: Illustration of the sets $N^=$, N^l and N^r with the physical representation, with 0 representing the depot/bin and the pairs $(2i-1, 2i)$, $i = 1, \ldots, n$ the rows. Plain arcs correspond to rows and dashed arcs denote some round trips.

Knowing that the depot is static and positioned on one side of the field, the vehicle harvests a row from the depot to cross the field and must harvest a row again to return to the depot without using diagonal arcs. Thus, if the number of rows is odd, there always will be a cycle remaining that forms either an outward or a return; it will be necessary to use a diagonal arc because no other row is available. We wish to generalize this characterization for instances with a high *Rmin* value and therefore to rows with a fixed direction.

Definition 7. A row that can be only harvested with a diagonal arc in a feasible solution, because it needs to be harvested in the good direction, is called a *limiting row*. Let us consider a *limiting set* $X \subseteq N$ as a subset of rows containing at least one *limiting row*.

To count the minimum number of limiting rows required in a feasible solution, it is needed to determine the maximum number, denoted by w, of rows that can be harvested in the wrong direction while still respecting the *Rmin* constraint. w is such that the following interval applies to *Rmin*:

$$q_{tot}(\mathcal{A}) - (w+1) * \delta \leq Rmin \leq q_{tot}(\mathcal{A}) - w * \delta \tag{4.1}$$

In equation 4.1, $q_{tot}(\mathcal{A}) = \sum_{(i,j)\in N} \max\{q_i(\mathcal{A}), q_j(\mathcal{A})\}$. $q_{tot}(\mathcal{A})$ represents the maximum quantity of $\mathcal{A}$-grapes effectively available for harvest, if all rows would be harvested in the good direction. In this same equation, δ denotes the amount of $\mathcal{A}$-grapes degraded during the latency issue for one $\mathcal{BA}$ transition. Notice that the more the value of *Rmin* tends to 100%, the smaller w is, forbidding more rows to be direction-free.

Lemma 3. *The number of rows harvested in the wrong direction, $n_{\overline{\tau}}$, is at most w and the number of rows harvested in the good direction, n_τ, is at least $|N^r \cup N^l| - w$.*

Proof. By definition of w, cf. Equation 4.1, it is clear that the allowed number of rows harvested in the wrong direction is at most w. Moreover, each row should be harvested

only once using then a unique direction; if a row is not harvested in the wrong direction, then it is harvested in the good direction. This applies to rows in $N^r \cup N^l$. So the number of rows in this set harvested in the good direction is at least $|N^r \cup N^l| - w$ (since rows of $N^=$ are always harvested in the good direction). □

Lemma 4. *Only one of the sets N^l and N^r can be limiting and the minimum (considering $w = 0$) number of limiting rows included in $N^l \cup N^r$ is given by $|n^l - n^r|$.*

Proof. The set $N^=$ cannot be a limiting set because all rows inside are harvested in the good direction. Note that N^l and N^r are complementary, i.e. round trips can be formed by harvesting one row from N^l and one row from N^r. Such round trips harvest both rows in the good direction. Note that there cannot be one row in the good direction and one row in the wrong direction because the good direction of one set is the wrong direction of the other. Moreover, making a round trip by harvesting both rows in the wrong direction would only increase the number of limiting rows. Thus, if $n^l > n^r$ then $n^l - n^r$ rows of N^l will be limiting otherwise $n^r - n^l$ rows of N^r will be limiting. Therefore, there will be, between the sets of rows N^l and N^r, $|n^l - n^r|$ limiting rows (at minimum, if $w = 0$). □

Lemma 5. *The w available wrong direction harvesting reduce the number of limiting rows by $2 * w$.*

Proof. Note that it is not worth collecting two rows in the wrong direction to create a round trip; it is always better to construct a round trip using one row in the good direction and one row in the wrong direction to maximize the number of round trips formed. Thus, it is possible to build w round trips and consequently to collect $2 * w$ limiting rows. □

Example 1. Consider Figure 7, if the *Rmin* value and corresponding constraint are such that $w = 1$ is obtained (i.e. one row that can be harvested in the wrong direction without violating the *Rmin* constraint). This available wrong direction row harvest allows for a round trip with all row extremities included in N^l (see Figure 7). The row $(9, 10)$ is harvested in the wrong direction allowing row $(11, 12)$ to be harvested in the good direction without taking diagonal arcs.

The complementarity of rows belonging to set N^l and N^r is illustrated with the cycle $(0, 16, 15, 13, 14, 0)$.

Lemma 6. *The (minimal, considering $w = 0$) number of limiting rows between $N^=$, N^l and N^r is given by $|n^l - n^r| - n^=$.*

Proof. Since rows of $N^=$ can be harvested in both directions, it is still possible to combine them with a row of N^s, $\forall s \in \{l, r\}$, to create a round trip by harvesting both rows in the good direction. Each of these round trips takes one row of $N^=$ and one row of N^s, $\forall s \in \{l, r\}$. Moreover, since only one of the sets N^l and N^r can contain limiting rows, cf. Lemma 4, round trips using rows from $N^=$ will only borrow rows from N^l or N^r but

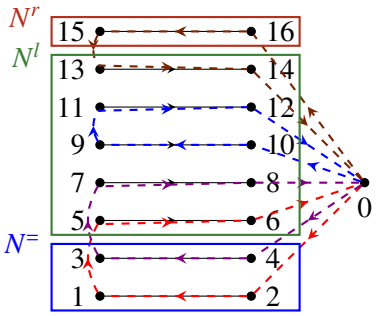


Figure 7: Illustration of example 1. Plain lines indicate good harvesting direction (no arrow if symmetrical). Dotted lines give direction chosen within the round trips.

not from both. Therefore, it follows the formula $|n^l - n^r| - n^=$ for the number of limiting rows between these three sets. □

Theorem 1. *The number of limiting rows, denoted by n_{lim}, in any feasible solution is at least $|n^l - n^r| - n^= - 2 * w$.*

Proof. The w available wrong direction harvesting allow reducing the number of limiting rows by $2 * w$, cf. Lemma 5. Thus, according to Lemma 6 the total number of limiting rows, and the number of diagonal arcs needed to cover them, is at least $|n^l - n^r| - n^= - 2 * w$. □

Theorem 1 gives a lower bound on the number of limiting rows in any feasible solution for a specific instance. This bound can be incorporated in the models in order to speed up their solving.

We exploit this bound on our compact formulation and VRPSolver model to enhance the linear relaxation by adding the following constraint (4.2). We were not able to find an efficient way of using this bound for the Constraint Programming and LocalSolver model.

$$\sum_{a \in \mathcal{D}} x_a^s \geq n_{lim} \tag{4.2}$$

5. Experimental results

The experiments were performed on an Optiplex 5055 computer with an AMD RYZEN 7 Pro 1700 (8 cores/4 Mo/16 Threads/3 GHz) processor and 32 Go RAM. CPLEX 12.8.0.0 was used for solving the problem based on the compact formulation model. For the column generation and branch-cut-and-price strategy (the VRPSolver oriented model), VRPSolver 0.4 [15] was used. Choco 4.10.1 [18] was used for constraint programming [24]. Local search was implemented using LocalSolver 10.0 [2]. All instances, implemented models and numerical results are available at [23].

5.1. Description of data sets

We have tested our optimization models on instances sets based on real data observed in a vineyard at INRA Pech-Rouge (Gruissan), located in southern France (cf. Figure 1).

Similarly to what was done in [8, 20, 24], the vine data set resulting from data of a real vine field was decomposed into a set of smaller vine data sets containing consecutive rows from the real data. The instances based on these generated yet realistic field data are denoted here *G*-instances.

The unique field data set with all the 24 rows was considered in this set of fields. The set also included the 2 field data sets that can be generated using 23 consecutive rows of the whole real field, as well as the 3 field data sets with 22 consecutive rows, and the 4 field data sets with 21 consecutive rows. To complement this set of 10 generated field data sets, a random selection of 5 generated field data set was made for decreasing numbers of rows, with a minimum of 10 consecutive rows per instance, yielding 55 supplementary generated field data sets. The set of fields that is used in this paper to form instances then comprises 65 field data sets. One instance results from the choice of generated field as data and from setting values for *Rmin* and *Cmax*. A class of instances comprises predetermined values for *Rmin* and *Cmax* and instances vary in a class by selecting one of the 65 generated field data sets as described above.

As for values of parameter *Cmax*, three cases were defined. The first case is a fixed value of 1000. The second case is a fixed value of 1500 (which is the hopper capacity of a real machine that had selective harvesting capabilities). The third case consists in choosing *Cmax* such that there are at least three routes in any solution of the specific instance. In this latter case, the value of *Cmax* depends on the quantities to harvest in the specific generated field data set.

For each instance, the *Rmin* parameter is a fixed value. This value can be chosen arbitrarily in range 0 to $q_{tot}(\mathcal{A})$ or as a percentage of $q_{tot}(\mathcal{A})$.

5.2. Numerical results

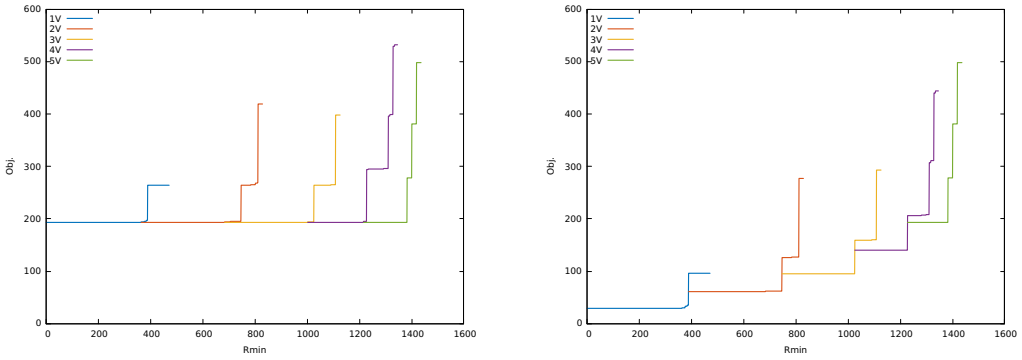
In this section, we describe the numerical experiments performed and their results.

First, the influence of the *Rmin* parameter on the solutions is studied. Then, the influence of the specific constraint alternate harvesting, which was present in the models provided [8, 20], is studied. Then, the benefits of using the notion of limiting rows are evaluated with regards to solving performance using VRPSolver model. Finally, the respective performances of our modelling approaches, with also some hybridizations (i.e. giving VRPSolver a first solution), are compared.

5.2.1. Influence of Rmin parameter

The aim of this section is to understand the behavior of the optimization with the variation of *Rmin* parameter. Our first hypothesis was that the *Rmin* parameter only influenced solutions when its value was high enough, forcing rows to be harvested in a certain direction to satisfy the *Rmin* constraint. However, studying the influence of *Rmin*

parameter on some instances also reveals that *Rmin* affects the optimization of selective routes (even when its value is small).



(a) On the total cost.

(b) On the selective cost.

Figure 8: The influence of the *Rmin* parameter, according to the number of selective routes belonging to solutions, for an 11-rows instance.

For this experiment, the number of selective vehicles (i-e the number of routes made by a selective vehicle) in a solution was fixed beforehand. The goal of the study is to maximize the amount of $\mathcal{A}$-grapes harvested while keeping operational cost minimal. We exactly solved (with VRPSolver) a real instance with 11 rows for each possible *Rmin* value and for each number of selective vehicles (one route per vehicle) between 1 and 5 (noted $1V$,...,$5V$ on the figures), having in total enough vehicles (routes) to harvest 100% of $\mathcal{A}$-grapes available, minimizing the harvesting cost, with a hopper capacity set to 500.

The Figure 8a shows the influence of *Rmin* parameter according to the number of selective routes fixed in solutions for the 11-rows instance. For the sake of clarity, we only show when plots have distinct values. Each line start means that it is better to use a new selective route than full hoppers to satisfy the *Rmin* constraint. Each line end (the upmost point of the line) is the cost corresponding to the highest reachable *Rmin* value for the corresponding number of selective vehicles.

To emphasize the *Rmin* parameter influence, notice that optimal solution for *Rmin* = 0% is 193 using five non-selective vehicles, and for *Rmin* = 100%, it is 498 using five selective vehicles, which represents a 155% increase! On Figure 8a, one can observe that optimal value for *Rmin* between 0 and 1384 (almost 97% of the total $\mathcal{A}$-grapes available) is always 193 using five vehicles (with the number of selective vehicles used increasing with the *Rmin* value, up to five). This figure exhibits *Rmin* thresholds, where the cost of the solution increases for a specific selective vehicle number; we focus our work on the last one (here for five selective vehicles) because it is the only one impacting the objective function. The figure also suggests that it is irrelevant to minimize the number of selective vehicles satisfying the *Rmin* constraint, as the optimal cost for most of the

Rmin value is the same, whereas the number of selective routes increases.

We also have performed the same experiment, with identical parameters and instance, but seeking only to satisfy the *Rmin* constraint, see Figure 8b. Notice that plots have the same shape as in Figure 8a; the only difference is the cost to harvest all rows. The selective routes cost increases with the *Rmin* value and the number of selective routes fixed in the solution. For a fixed number of selective routes in the solution, the selective cost considerably rises when maximizing the $\mathcal{A}$-grapes harvested, therefore it is better to take another route when this threshold is exceeded.

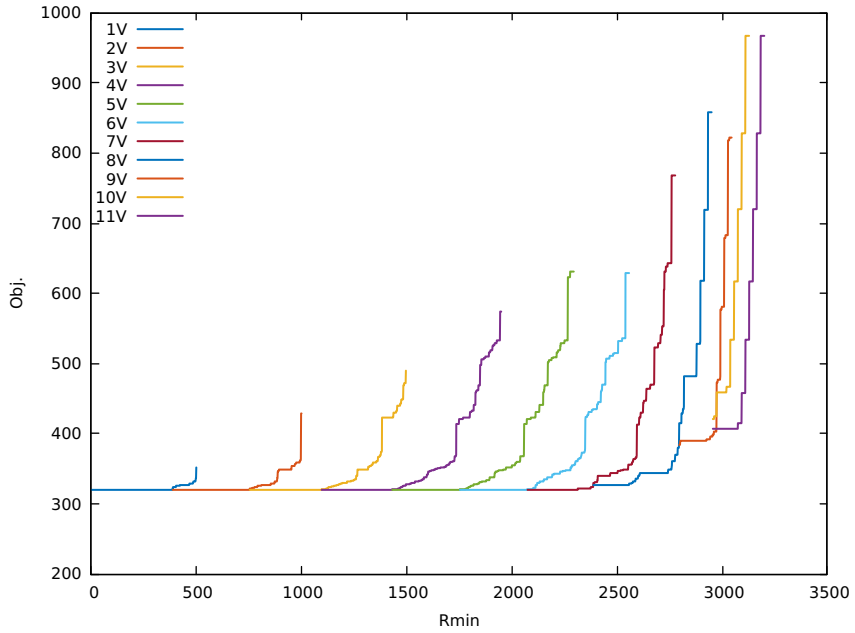


Figure 9: The influence of the Rmin parameter according to the number of selective routes belonging to solutions for the 24 rows instance.

These experiments demonstrate the complexity brought by the *Rmin* constraint on the optimization. Determining if using k selective vehicles rather than $k + 1$ and compensate the cost with the non-selective routes harden the resolution of the problem when the *Rmin* value is close to the selective vehicle's thresholds. We also have performed the same trials on various instances and observed the same behavior, see Figure 9 for the thresholds results on the 24-rows instance.

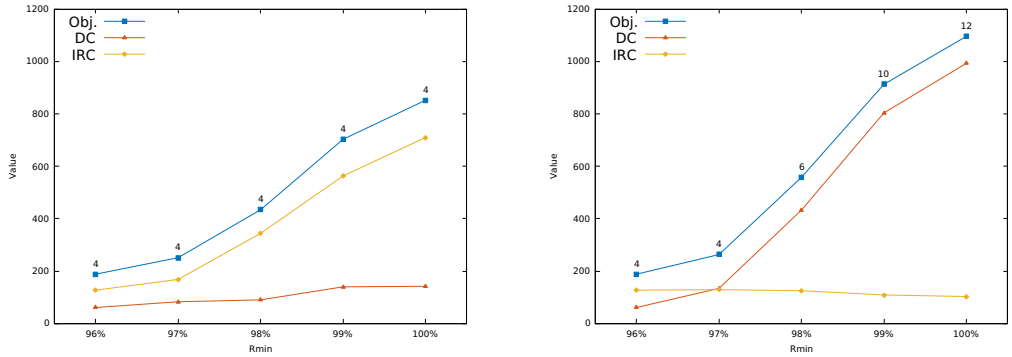
5.2.2. Influence of alternate harvesting

In this section, we empirically show that alternate harvesting (AH) restricts the initial problem and leads to chimerical solutions when *Rmin* value increases.

The depot cost (DC) corresponds to the cost of the arcs connected to the depot in a solution, and the inter-row cost (IRC) is the cost of arcs between rows.

We compare objective function value, depot cost and inter-row cost for optimal solution of the 24-rows instance (found with VRPSolver) with a *Cmax* value of 1500 and a *Rmin* threshold oscillating between 0% and 100% (with a focus around 100%) of total $\mathcal{A}$-grapes available in the field with and without diagonal arcs.

The Figure 10 gives the results of this experiment. The labels on the points of the plots indicate to the number of routes used in the solution. As was shown already in Section 5.2.1, the *Rmin* value does not influence the objective function that much between 0% and 96% (it is rather the number of selective routes that augments). The objective value rises for the higher values of *Rmin* because rows tend to be harvested in a unique direction with a diagonal arc. The objective function value increases drastically from 181 for *Rmin* = 0% to 852 for *Rmin* = 100% with the non-alternate harvesting (nAH) and 1096 for the AH representing a ratio of 4.7 for the nAH and 6.05 for the AH.



(a) With nAH. (b) With AH.

Figure 10: Evolution of objective value, depot cost and inter row cost according to *Rmin* value increase.

One can see that the number of routes needed in the alternate harvesting (AH) increases considerably for the *Rmin* values close to 100% due to the forced direction changes and the return to the depot. In this instance, solutions use at most three times more routes in AH (twelve routes needed for 100% *Rmin*) than in non-alternate harvesting (nAH) (four routes). This increase of the number of routes indicates that for the majority of routes, the machine is harvesting a unique row within each route. Thus the DC grows significantly whereas the IRC stays constant. However, for the nAH the number of routes used in the solution does not increase; therefore, this is the IRC that rises substantially, and the DC remains stable.

The first difference in the objective function value happens for a *Rmin* value of 97%, for the AH the value is 263 and for the nAH it is 251. This is due to the choice for diagonal arc used to cross the field in the nAH instead of returning to the depot for the AH. With the nAH, the diagonal arcs used to harvest *limiting rows* can be optimized, and consequently a smaller objective function value can be obtained. For the *Rmin* value of 100% the objective value is 852 for nAH and 1096 for AH, which represents a benefit of

28%.

5.2.3. Influence of limiting rows

In Section 4, a lower bound on the number of limiting rows in a solution, and thus on the number of diagonal arcs needed to harvest them, was found and provided in Theorem 1. The purpose of this section is to exhibit the impact of Theorem 1 on the optimization using linear programming. More specifically, the benefits of adding the valid inequality corresponding to this lower bound in the model using VRPSolver will be studied.

For this experiment, let us introduce twelve classes of instances decomposed into two sets of instances (of six classes): one with $Cmax = 1000$ and the other with $Cmax = 1500$. For each value of $Cmax$ let us consider $Rmin = \{95\%,96\%,97\%,98\%,99\%,100\%\}$. The twelve classes have the same 65 instances randomly selected. All the instances were solved with and without the constraint (4.2) with a resolution time limit fixed at 600 seconds using VRPSolver.

The Table 1 summarizes the results, comparing the integrality gap for the root node and the average resolution time (optimal integer solution) for all instances of each class and calculating the ratio between the values obtained with and without constraint (4.2).

	$Cmax = 1000$					
	without (4.2)		with (4.2)			
$Rmin$ (%)	gap(%)	time(s)	gap(%)	time(s)	gap	time
95	2,56	67,07	1,35	12,74	$*1,89$	$*5,26$
96	8,78	201,58	1,15	6,50	$*7,65$	$*31,00$
97	14,43	307,65	0,73	5,12	$*19,77$	$*60,03$
98	13,26	321,83	0,59	4,86	$*22,52$	$*66,26$
99	10,20	374,97	0,25	4,76	$*41,16$	$*78,79$
100	0,29	3,81	0,12	2,44	$*2,15$	$*1,56$
	$Cmax = 1500$					
95	3,00	160,51	1,42	75,51	$*2,11$	$*2,13$
96	9,89	331,75	1,01	42,80	$*9,79$	$*7,75$
97	16,25	470,35	0,62	26,40	$*26,05$	$*17,82$
98	16,38	525,81	0,37	22,82	$*44,55$	$*23,04$
99	11,28	465,02	0,30	33,23	$*37,69$	$*13,99$
100	0,19	36,20	0,17	17,84	$*1,08$	$*2,03$

Table 1: Synthesis of the results on the influence of the constraint (4.2) on the linear relaxation of the root node of the column generation using VRPSolver.

Let us notice first of all that without the constraint (4.2), the instances are increasingly difficult to solve when the value of $Rmin$ grows, except for $Rmin = 100\%$ which is a particular case.

Indeed, the average of integrality gaps rises considerably when $Rmin = 95\%$ increases, it goes from 2.56% for $Rmin = 95\%$ to 14.43 for $Rmin = 97\%$ and to 10.20% for $Rmin = 99\%$ with $Cmax = 1000$ (before reducing drastically for $Rmin = 100\%$ reaching an average value of 0.29%). This results in a high slow down of the resolution

times, on average 67.07 seconds for *Rmin* = 95% to 307.65 seconds for *Rmin* = 97% with *Cmax* = 1000.

Thus the number of instances optimally solved in less than 600 seconds reduces, except for *Rmin* = 100% where the average time to solve the instances is 3.81 seconds with *Cmax* = 1000, and 36.20 seconds with *Cmax* = 1500.

In comparison, when the constraint (4.2) is added, the opposite effect occurs: the average of the integrality gaps decreases when *Rmin* grows (from 1.35% for *Rmin* = 95% to 0.12% for *Rmin* = 100% with *Cmax* = 1000). Therefore the average of resolution times reduces too (from 12.74 seconds on average for *Rmin* = 95% to 2.44 seconds for *Rmin* = 100% with *Cmax* = 1000). Thus, all the instances are optimally solved in a few seconds.

The results, for the classes of instances with *Cmax* = 1500, follow the same logic, but with a slightly less significant effect (for the resolution times and not for the reinforcement of the linear relaxation) because these instances are harder to solve because of the increase in the size of the routes.

For instance classes with *Rmin* = 100%, the gap averages are naturally quite close to 0 because the *Rmin* constraint naturally forces all the rows to be harvested in the good direction.

For classes of instances with *Rmin* = 95%, the ratio of gaps and resolution times is not so representative because few instances have a *Rmin* value exceeding the value last *Rmin* threshold (see Section 5.2.1). On the other hand, results are more significant for classes with $Rmin \in \{96\%, \ldots, 99\%\}$ where the gains are remarkable (at most a factor of 44.55 for the gaps and 78.79 for the resolution times).

However, one may observe that the added constraint improves the performances of VRPSolver even when it does not especially strengthen the linear relaxation. For example, for instance class G-100-1500, the ratio for the gap is only $*1.08$, thus quite small, yet the resolution time is reduced by a factor of 2.03 on average.

On average, for all the instances tested, the gain factor for the integrality gap is $*18.06$ and for the resolution time is $*25.80$, which represents a considerable increase in the model's performance with this single additional constraint.

The constraint (4.2) adds information on the number of diagonal arcs required in a solution. Therefore, this constraint enhances the linear relaxation of the nodes of the Branch-Cut-and-Price tree, thus cutting off the exploration of more branching nodes and consequently converging faster to an optimal solution.

5.2.4. The performance results

This section provides a performance comparison between the approach using VRPSolver, our other approaches, and hybridization approaches based on VRPSolver. By hybridization, we mean here providing an initial solution rapidly computed as an upper bound to VRPSolver helping to prove the optimality of solutions. For the sake of simplicity, let us note:

- the COMPACT formulation "ILP ";
- the CONSTRAINT PROGRAMMING model "CP ";
- the LOCALSOLVER model "LS ";
- the VRPSOLVER model "VS ".

For non-hybrid methods, the resolution time limit was fixed at 600 seconds. For hybrid methods, VS+X with X being the methods ILP, CP and LS, a solution found in 60 seconds with model X was given as the first solution (upper bound) for VRPSOLVER with a time limit of 540 seconds.

The parameter *Rmin* was fixed to 95% of the total $\mathcal{A}$-grapes available. This value was chosen because preliminary studies showed that the objective function (harvesting time) was nearly the same when considering lesser values. Three classes of instances G-3*R*-95, G-1000-95 and G-1500-95, defined according to section 5.2, will be considered.

Table 2 synthesises instances and parameter values.

	#rows	*Cmax*	*Rmin*	#instances
G-1000-95	$10 \leq n \leq 24$	1000	95%	65
G-1500-95	$10 \leq n \leq 24$	1500	95%	65
G-3*R*-95	$10 \leq n \leq 24$	[420, 1333]	95%	65

Table 2: Parameters variation for each instance class

Please note that, in the instance class labelling G-X-Y, *X* denotes the choice for *Cmax* (1000, 1500, 3*R*) and *Y* denotes the choice for *Rmin*.

For each class of instances, we compare the percentage of optimal solutions (normalized distance equals to 0%) found and the percentage of solutions with small but significant normalized distances to the optimal solution (5% and 10%) at a given time.

The Table 3 summarizes the results of this experiment at the end of the time limit. Figures 11 and 12 show the performance plots for the G-1500-95 instance class. The legend of the figures follows the performance of the plots in increasing order (top legend has the better performance). The performance of methods for classes G-3*R*-95 and G-1000-95 follow the same behaviors as for class G-1500-95. Yet, because the instances are easier to solve, the corresponding graphics are more stacked and are not provided in this paper.

It can be observed that optimal solutions are, comparatively, not easily obtained with the models ILP, LS and CP. Yet, many instances can be solved using these models with a distance of at most 10% from optimal solution (almost 100% for ILP, 90% for LS, and 60% for CP).

VRPSOLVER clearly outperforms the other methods even without hybridization, notably because it handles very powerful optimization techniques.

class	ϵ	ILP	LS	CP	VS	VS + LS	VS+CP	VS+ILP
G-3*R*-95	0%	46,15%	29,23%	15,38%	100%	100%	100%	100%
	5%	95,38%	66,15%	23,08%	100%	100%	100%	100%
	10%	98,46%	83,08%	32,31%	100%	100%	100%	100%
G-1000-95	0%	50,77%	24,62%	24,62%	100%	100%	100%	100%
	5%	95,38%	80,00%	49,23%	100%	100%	100%	100%
	10%	100%	89,23%	58,46%	100%	100%	100%	100%
G-1500-95	0%	63,08%	15,38%	29,23%	93,85%	100%	98,46%	100%
	5%	100%	55,38%	38,46%	93,85%	100%	98,46%	100%
	10%	100%	78,46%	60,00%	93,85%	100%	98,46%	100%

Table 3: Synthesis of performance results for DHP models at the end of the time limit (600 seconds), each value corresponds to the number of optimal solutions found in percentage.

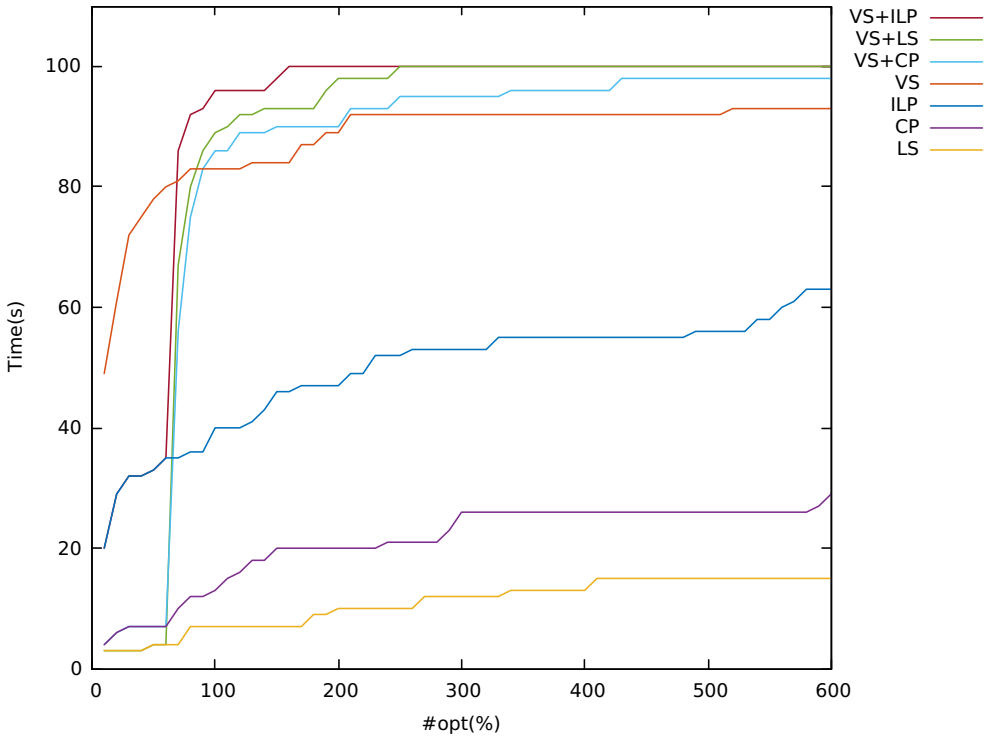


Figure 11: Evolution of the number of optimal solutions according to the resolution time for the G-1500-95 class.

When hybridizing, the intention with methods LS and CP is to obtain good solutions to feed the initial solution (upper bound) for VS. It can be observed that the number of good solutions ($< 10\%$) obtained by LS increases proportionally with resolution time. However, for the local search approach, these results remain of lower quality than with ILP.

Whether used to obtain directly optimal solutions or to get good solutions quickly, our CP model seems not to be efficient in comparison with ILP for the sake of finding optimal solutions and with LS for the sake of getting good solutions quickly. The main difficulty of the CP model is that the variables and constraints are too free (because of the asymmetric harvesting and thus the harvesting directions) and thus propagate badly during the search.

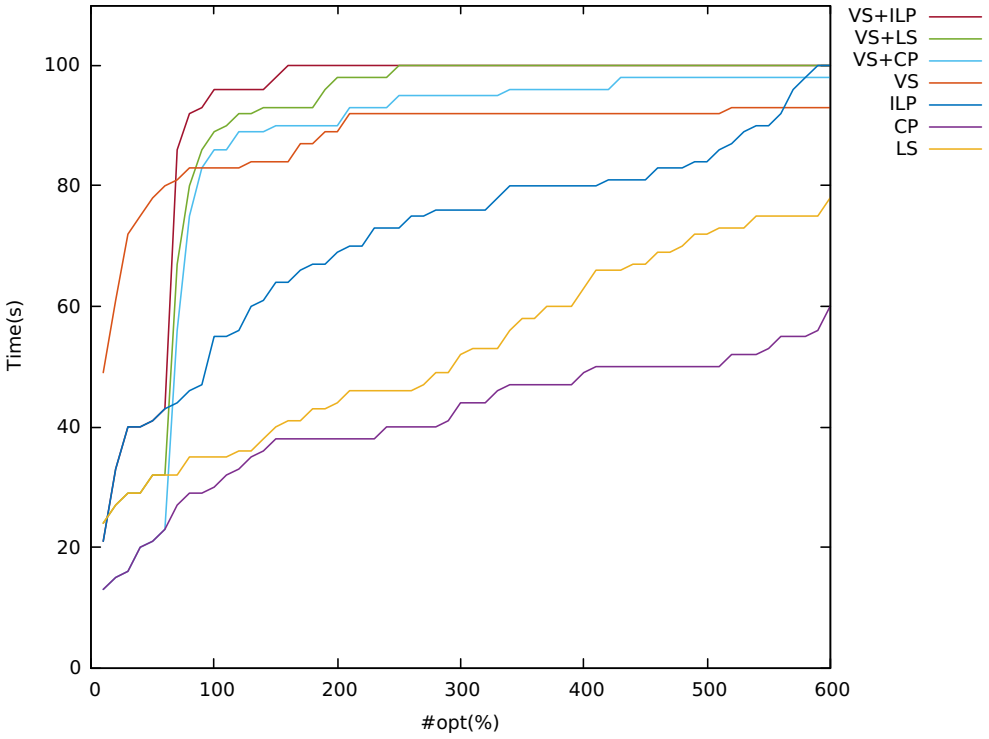


Figure 12: Evolution of the number of solutions with distance of at most 10% of the optimal value according to the resolution time for the G-1500-95 class.

The compact formulation was designed to compute optimal solutions. In 600 seconds, the optimal solution is found, for each class of instances, in at most 63.08% of the cases (see Table 3). It appears that, in the worst case, G-3R-95, it also finds a solution with a distance of 10% from the optimal value in 98.46% of the class instances. For same class, this is better than the LS method (83.08%) and than the CP method (32.31%). For

other classes, it finds such approximate solution in the range of 10% from the optimal value in 100% of instances. So, ILP was found to be a good candidate for hybridization with VS.

The hybrid method behaves in the same way (finds all optimal solution) as VS alone for almost all classes of instances, with nevertheless a small and significant improvement for the G-1500-95 class (going from 93.85% for VS to 98.46% for VS +CP and to 100% with VS +ILP and VS +LS).

Nevertheless, in Figure 11 one can see the influence of hybridization scheme on time used to find and prove optimal solution with VS. For example, in the most difficult class of instances G-1500-95, the hybrid method VS+ILP finds the optimal solution for 95% of the instances in at most 70 seconds, instead of 80% for VS alone while the first 60 seconds of the hybrid method uses only ILP.

6. Conclusion

This paper contributes in investigating usability and efficiency of generic solvers to solve specific and practical problems. The Differential Harvest Problem comes from a precision farming perspective and we investigated it as a heterogeneous vehicle routing problem variant. For the Differential Harvest Problem, the key point was to understand the problem-specific parameters and constraints. We tackled new Differential Harvest Problem variants. We also proposed new bounds for instances for which the *Rmin* value is high, which were not investigated until now.

Many parameters influence the instance difficulty, such as the *Rmin* value, the hopper capacity *Cmax*, and the 2-DVPP parameters such as the distribution of $\mathcal{A}$-grapes and $\mathcal{B}$-grapes in rows. A bundle of methods, exact and heuristic, were addressed and analyzed with experiments on real and simulated instance classes. VRPSolver works better with good upper bounds. The results in Table 3, with ILP performing slightly better than LS in hybridization, suggest that future work needs to be done designing specific and efficient heuristic algorithms or enhancing the LS model. Already, thanks to VRPSolver and hybridization, and for the first time, a 24-rows real instance could be solved in less than 60 seconds. This is a major improvement with regards to previous results (12 rows in around 118 seconds in [8] and 14 rows in 441 seconds in [20]).

However, mathematically, the current state of this work has limitations. We were not able to reach scalability on some simulated instances with 50 rows. These simulated instances, which were purposely designed to exhibit the difficulties of the problem, were not solved optimally within a one hour time limit (which is a usual time limit for testing exact methods for 2-DVPP).

From a practitioner point of view, the developed method in its current state is largely able to solve the problem for most of the french vine fields, considering their usual size and number of rows, provided quality data and harvesting machinery with sorting capabilities are available. Realistic and most probable vineyard cases should be expected

to be solved in a few seconds, due to the fact that *Rmin* should often be less than 95%. The "harder" vineyard cases with a number of rows typically less than 25 rows should be solved in a few minutes, as was shown in this paper.

Acknowledgements

This work was supported by the French National Research Agency under the Investments for the Future Program, referred as ANR-16-CONV-0004 We also thank the reviewers for their useful comments helping to improve the structure and the quality of the paper.

References

[1] R. Baldacci, M. Battarra, and D. Vigo. Routing a heterogeneous fleet of vehicles. In *The vehicle routing problem: latest advances and new challenges*, pages 3–27. Springer, 2008.

[2] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua. Localsolver 1. x: a black-box local-search solver for 0-1 programming. *4or*, 9(3):299–316, 2011.

[3] D.D. Bochtis and C. G Sørensen. The vehicle routing problem in field logistics part i. *Biosystems engineering*, 104(4):447–457, 2009.

[4] D.D. Bochtis and C.G. Sørensen. The vehicle routing problem in field logistics: Part ii. *Biosystems engineering*, 105(2):180–188, 2010.

[5] Z. Borcinova. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review*, pages 463–469, 2017.

[6] Eric Bourreau, Matthieu Gondran, Philippe Lacomme, and Marina Vinot. *De la programmation linéaire à la programmation par contraintes*. Ellipses, 2019.

[7] N. Briot, C. Bessiere, B. Tisseyre, and P. Vismara. Integration of operational constraints to optimize differential harvest in viticulture. In *Precision agriculture'15*, pages 111–129. Wageningen Academic Publishers, 2015.

[8] N. Briot, C. Bessiere, and P. Vismara. A constraint-based approach to the differential harvest problem. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 541–556, Cham, 2015. Springer International Publishing.

[9] M. Buljubašić and M. Vasquez. Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research*, 76:12–21, 2016.

[10] M. Burger, M. Huiskamp, and T. Keviczky. Complete field coverage as a multi-vehicle routing problem. *IFAC Proceedings Volumes*, 46(18):97–102, 2013.

[11] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231–262, 2001.

[12] C. Contardo and R. Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 2014.

[13] L. M. R dos Santos, A. M Costa, M. N Arenales, and R. H. S Santos. Sustainable vegetable crop supply problem. *European Journal of Operational Research*, 204(3):639–647, 2010.

[14] E. C. man Jr, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[15] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, 2020.

[16] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Solving bin packing problems using vrpsolver models. In *Operations Research Forum*, volume 2, pages 1–25. Springer, 2021.

[17] M. Plessen. Partial field coverage based on two path planning patterns. *Biosystems engineering*, 171:16–29, 2018.

[18] C. Prud'homme, J.-G. Fages, and X. Lorca. Choco documentation. *TASC, INRIA Rennes, LINA CNRS UMR*, 6241:64–70, 2014.

[19] R. Saddem-Yagoubi, O. Naud, K. G. Dejean, and D. Crestani. New approach for differential harvest problem: The model checking way. *IFAC-PapersOnLine*, 51(7):57–63, 2018.

[20] R. Saddem-Yagoubi, O. Naud, K. Godary-Dejean, and D. Crestani. Model-checking precision agriculture logistics: the case of the differential harvest. *Discrete Event Dynamic Systems*, 30:579–604, 2020.

[21] H. Seyyedhasani and J. Dvorak. Reducing field work time using fleet routing optimization. *Biosystems engineering*, 169:1–10, 2018.

[22] F. CR Spieksma. A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers & operations research*, 21(1):19–25, 1994.

[23] G. Volte. https://doi.org/10.5281/zenodo.6108672, February 2022.

[24] G. Volte, E. Bourreau, R. Giroudeau, and O. Naud. Exact method approaches for the differential

harvest problem. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 492–510. Springer, 2020.