



UNIVERSITÀ POLITECNICA DELLE MARCHE  
Repository ISTITUZIONALE

## A MIP-Based Heuristic for Pickup and Delivery on Rectilinear Layout

This is the peer reviewed version of the following article:

*Original*

A MIP-Based Heuristic for Pickup and Delivery on Rectilinear Layout / Arbib, C., Pizzuti, A., Ranjbar, F.K., Smriglio, S. - 1623:(2022), pp. 211-226. (9th International Conference on Operations Research and Enterprise Systems, ICORES 2020, and 10th International Conference on Operations Research and Enterprise Systems, ICORES 2021 Valletta, online 22 - 24 february 2020, 4-6 February 2021) [10.1007/978-3-031-10725-2\_11].

*Availability:*

This version is available at: 11566/325673 since: 2024-01-02T17:32:29Z

*Publisher:*

Springer Science and Business Media Deutschland GmbH

*Published*

DOI:10.1007/978-3-031-10725-2\_11

*Terms of use:*

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. The use of copyrighted works requires the consent of the rights' holder (author or publisher). Works made available under a Creative Commons license or a Publisher's custom-made license can be used according to the terms and conditions contained therein. See editor's website for further information and terms and conditions.

This item was downloaded from IRIS Università Politecnica delle Marche (<https://iris.univpm.it>). When citing, please refer to the published version.

*Publisher copyright:*

Springer (conference paper) - Postprint/Author's accepted Manuscript

This version of the conference paper has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use <https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms>, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: 10.1007/978-3-031-10725-2\_11.

(Article begins on next page)

# A MIP-based heuristic for pickup and delivery on rectilinear layout

Claudio Arbib<sup>1</sup>[0000–0002–0866–3795], Andrea Pizzuti<sup>2</sup>[0000–0003–3255–8378],  
Fatemeh K. Ranjbar<sup>1</sup>[0000–0002–0948–1440], and Stefano  
Smriglio<sup>1</sup>[0000–0001–8152–003X]

<sup>1</sup> DISIM, Università degli Studi dell’Aquila, L’Aquila, Italy  
{claudio.arbib,stefano.smriglio}@univaq.it,  
fatemeh.kafashranjbar@graduate.univaq.it  
<sup>2</sup> DII, Università Politecnica delle Marche, Ancona, Italy  
a.pizzuti@univpm.it

**Abstract.** The Vehicle Routing Problem with Pickups and Deliveries (VRPPD) arises in many application contexts and has been intensively studied in the last decades. We investigate the special case where pickup and delivery locations are distributed on a line. Although this situation is frequent when handling material in manufacturing systems with rectilinear layout, this case has not received enough attention so far. Derived from a real application, our general model also features load/unload times, vehicle capacities and the absence of a depot. A two-stage MIP-based heuristic that exploits such a special topology is devised, and its performance is assessed within an industrial case study provided by a large semiconductor manufacturer. We first compare our method to a standard Clarke and Wright type heuristic, then document its practical impact when implemented in a dynamic environment.

**Keywords:** Vehicle routing · Pickup and delivery · Mixed-integer programming

## 1 Introduction

We are given a set  $I$  of  $n$  locations placed on a line at fixed positions and a set  $R$  of requests, each of which, say  $r \in R$ , asks for handling an amount  $l_r$  of commodity from a *pickup location*  $p_r \in I$  to a *delivery location*  $d_r \in I$  (without intermediate transshipments at other locations). Handling is carried out by a set  $K$  of  $m$  vehicles. Each vehicle  $k \in K$  travels at constant speed  $v$  and has a limited loading capacity  $c_k$ . Finally, for a request  $r \in R$ , pickup/delivery operations take  $u \cdot l_r$  time, where  $u$  is a given constant. We wish to determine vehicle routes to process all the requests while minimizing the *makespan*, that is, the largest completion time among all requests.

This problem is a special case of the classical VEHICLE ROUTING PROBLEM WITH PICKUPS AND DELIVERIES (VRPPD), intensively studied in the last decades (for transportation services dedicated to people, one speaks of a DIAL-A-RIDE

PROBLEM). We refer the reader to [Bereglia et al. 2007, Vidal et al., 2020] for an introduction to this huge body of research.

A distinguishing feature of our problem is the rectilinear layout of the pickup/delivery locations: so we will call it the RECTILINEAR-VRPPD (R-VRPPD). Such a situation arises, for instance, in manufacturing plants where machining facilities are deployed along an aisle. This case has not received enough attention in the VRP literature; yet, the rectilinear layout has been investigated in related studies. [Guan, 1998] considers the problem of transporting a set of items, each with a specific origin and destination, along a path; the Author observes that minimizing the tour length is NP-hard if items cannot be dropped at intermediate stations, while it can be solved in linear time otherwise. [Wang et al. 2006] propose polynomial-time algorithms to solve a single-commodity pick-up and delivery problem on a path using a single vehicle with unit, or general finite, or unlimited capacity. Both studies do not capture the complexity of our case, where a fleet of vehicles is given, each requests involves a separate commodity, and items cannot be dropped at intermediate stations.

Another feature of our model is the way load/unload is considered. Such operation has in fact a double effect on solution, in terms of both capacity and time usage. While vehicle capacity is addressed in various papers (one for all, [Ralphs et al., 2003]), load/unload time is generally dealt with by incorporating it into the total time required to fulfill a request. However, an economy of scale can be gained by suitably grouping requests that share the origin and/or the destination (provided that enough capacity is available). To the best of our knowledge, both issues are not addressed simultaneously in the literature.

A third feature of our model is that it is designed for use in a dynamical environment, where the initial position of each vehicle can be arbitrary in the aisle. In fact, in the industrial case which stimulated this study [Arbib et al. 2021], the production process is continuous and no location can a-priori be identified as the vehicle depot. Typical solution methods for the VRPPD are instead developed under the assumption of a single depot, from which vehicles start at the beginning of the work shift and where they go back at its end. This not irrelevant detail makes, for instance, standard *cluster-first route-second* heuristics not directly usable, as the cost of assigning a cluster to a vehicle is not independent on the vehicle but instead depends on its initial position.

Summarizing, our contribution is twofold: on one hand we focus on practical aspects (topology, load/unload operations, absence of a depot) not satisfactorily addressed in the VRP literature; in so doing, on the other hand, we propose a way to exploit such peculiarities and gain in both algorithm efficiency and solution accuracy. In particular, the rectilinear layout allows us to devise a math-heuristic that tackles the problem in two stages, according to the following *cluster-first route-second* strategy:

1. **Request Assignment (Clustering).** Via a MIXED INTEGER (LINEAR) PROGRAM (MIP), assign the requests in  $R$  to as many distinct segments of the line, as available vehicles. Each segment will identify a route to be next assigned to some vehicle.

2. **Vehicle Assignment and Scheduling (Routing).** Schedule each vehicle on each distinct route and determine in this way the time required of the vehicle to fulfill all the requests assigned to that route. Then, by solving a BOTTLENECK MATCHING PROBLEM, find a one-to-one vehicle-to-route assignment that minimizes the largest among all vehicles completion time.

The whole heuristic is illustrated in §2. Its performance is investigated in §3 by an industrial case study provided by a large semiconductor manufacturer, first illustrated in [Arbib et al. 2021], which motivated the present study. Here we address the RVRPPD problem along with the proposed decomposition framework from a more general perspective. In particular, we document a comparison of our heuristic to a standard Clarke and Wright’s *cluster-first route-second* heuristic for the VRP (adequately modified to match the problem features inherited from the application). This is done in a *static* environment, where all requests are known in advance. Then, we review the adaptation of the heuristic to a dynamic context, as in [Arbib et al. 2021]: now the problem becomes an R-VRPPD with time windows, as requests are released at different time instants, and no specified depot, as the problem is solved with arbitrary initial distributions of vehicles. On the whole, in §3.3, a meaningful practical benefit to the industrial process is documented.

## 2 MIP-based heuristic

In this section we recall the MIP-heuristic developed in [Arbib et al. 2021]. The rectilinear topology allows us to represent the locations as a set of integers  $I = \{i_1, \dots, i_n\}$ , corresponding to the location coordinates along the line. Now, given a subset of requests  $P \subseteq R$ , we define a minimal interval  $[a(P), b(P)]$  that spans between the leftmost and rightmost pickup/delivery location involved in  $P$ ; formally

$$a(P) = \min_{r \in P} \{p_r, d_r\} \quad b(P) = \max_{r \in P} \{p_r, d_r\}$$

To illustrate the above operation, let us consider the following

*Example 1.* Let  $I = \{1, 2, 3, 4, 5, 6, 13\}$  and  $R$  as in Table 1

**Table 1.** Data for Example 1.

| Request | $p_r$ | $d_r$ | $l_r$ |
|---------|-------|-------|-------|
| 1       | 5     | 1     | 1     |
| 2       | 1     | 3     | 2     |
| 3       | 13    | 6     | 2     |
| 4       | 5     | 14    | 3     |
| 5       | 4     | 3     | 1     |

Assuming  $P = \{1, 4, 5\}$ , stations  $d_1 = 2, p_4 = 5, d_5 = 3$  (stations  $p_1 = 5, d_4 = 14, p_5 = 4$ ) are the leftmost (rightmost) involved in those requests; hence,  $a(P) = 2, b(P) = 14$ .

We call any such interval a *span* of  $P$ . Spans are no more than the pairs of stations involved in  $R$ , namely  $\leq \frac{q(q-1)}{2} = O(q^2)$  with  $q = \min\{n, 2|R|\}$ . Let  $S$  indicate the set of all those spans, and denote by  $w^s = b^s - a^s = b(P) - a(P)$  the width of the  $s \in S$  corresponding to  $P \subseteq R$ . In our example  $n = 14$  implies 91 possible spans in the worst case and the span  $s = [2, 14]$  has  $w^s = 14 - 2 = 12$ .

Our two-stage heuristic for the R-VRPPD is based on the following decomposition principle:

1. Assign each request in  $R$  to a span in a set  $S^* \subseteq S$  with  $|S^*| = |K| = m$ ;
2. Assign a vehicle  $k \in K$  to each  $s \in S^*$  and schedule its requests.

In stage 1, we cluster requests into spans of convenient width — the larger the span the fewer the requests — in order to balance the estimated operation time (travel + load/unload). In stage 2, we determine elementary schedules where each vehicle “sweeps” from end to end the span it is assigned to, changing direction a minimum amount of times. Details on the two stages are given in next sections.

An important property of this decomposition is that a solution can easily be reconfigured at run time against possible vehicle capacity reduction, as well as urgent deliveries or other exceptions. This property will be fruitfully exploited when applying the heuristic in a dynamic environment (see §3.3).

## 2.1 Stage 1: span selection and request assignment

To simplify the method description, let us for the moment make a rough approximation (to be refined later) of the time a vehicle would take to cover a span  $s$ . We consider

- (i) the span width  $w^s$ : the larger the  $w^s$ , the longer will expectedly be the transfer time needed to cover its requests.
- (ii) the overall volume  $l^s$  of requests assigned to  $s$ : large  $l^s$  means long time spent in load/unload operations;

While  $w^s$  only depends on  $s$ , the volume  $l^s$  depends on the requests assigned to  $s$ . Let  $R^s \subseteq R$  denote the set of requests whose pick-up and delivery stations fall within the ends of  $s$ . Then:

$$l^s = \sum_{r \in R^s} l_r x_r^s$$

where  $x_r^s = 1$  if  $r$  is assigned to  $s$  and 0 otherwise. Using these variables, plus the activation variables  $y^s \in \{0, 1\}$  with  $y^s = 1$  if and only if  $s \in S$  is chosen, we can select the  $m$  spans via the following MIP (see also [Arbib et al. 2021]):

$$\begin{aligned}
& \min_{z \in \mathbb{R}} z & (1) \\
& \text{s.t.} \\
& z - \frac{w^s}{v} y^s - 2u \sum_{r \in R^s} l_r x_r^s \geq 0 & s \in S \\
& \sum_{s \in S_r} x_r^s = 1 & r \in R \\
& x_r^s \leq y^s & s \in S, r \in R^s \\
& \sum_{s \in S} y^s \leq m \\
& \sum_{r \in R^s} l_r x_r^s \leq c & s \in S \\
& x_r^s, y^s \in \{0, 1\} & s \in S, r \in R^s
\end{aligned}$$

where  $S_r \subseteq S$  contains all the spans that can accommodate request  $r$ , and  $u, v$  are the nominal load/unload time and vehicle speed, respectively. Coupled with the objective function, the first set of constraints defines  $z$  as the largest among the the completion time estimations of the requests attributed to a span. The second set of constraints enforces that every requests is assigned to one span. The third selects a span when assigned a request. The remaining constraints ensure that the spans selected do not exceed available vehicles, and that vehicle capacity is in turn never exceeded.

The first four lines of program (1) have the aspect of an  $m$ -centre model, where spans corresponds to facilities, or centroids, and requests to demand points. In brief, an  $m$ -centre is a set  $S^*$  of  $m$  points (called centroids) that minimizes the largest distance between any demand point and the closest point in  $S^*$ : for a survey on  $m$ -centre problems see e.g. [Calik et al., 2015]. Our model differs from this problem in the distance function, that includes centroid costs (span lenghts) and, as said, for the additional knapsack constraints (vehicle capacities).

Once solved, program (1) returns:

- a set  $S^*$  of  $m$  spans;
- a partition  $\{R^s : s \in S^*\}$  of the requests set to be assigned to vehicles in the subsequent stage 2;
- a lower bound  $z^*$  to the minimum time necessary to schedule all the requests in  $R$ .

Let us now refine the initial estimation of transfer time, that we assumed proportional to the span width  $w^s$ : this estimate is in fact reasonable as far as the vehicle covers the span in one direction only, but becomes poor if it has to go back and forth. To obtain a better estimate, modify program (1) by distinguishing the requests in  $R^s$  according to whether they require a transfer from left

to right (forward requests,  $F_t^s$ ) or from right to left (backward requests,  $B^s$ ). Precisely

$$F^s = \{r \in R^s : p_r < d_r\} \quad B^s = \{r \in R^s : p_r > d_r\}$$

We then add 0-1 variables  $f^s, b^s$ , subject to

$$\begin{aligned} x_r^s &\leq f^s \leq y^s & r \in F^s \\ x_r^s &\leq b^s \leq y^s & r \in B^s \end{aligned} \quad (2)$$

for any  $s \in S$ . That is, span  $s$  is used if a vehicle uses it either in forward or in backward direction (or both); conversely, if not used in forward (backward) direction, the span cannot be assigned any forward (backward) request. The constraint of model (1) that ensures to use  $\leq m$  spans is maintained, but the first set of inequalities, defining the max completion time, is replaced by

$$z - \frac{w^s(f^s + b^s)}{v} - 2u \sum_{r \in R^s} l_r x_r^s \geq 0 \quad (3)$$

for any candidate span  $s \in S$ . Thus, if the span is assigned requests in one direction only, the estimated travel time is proportional to  $w^s$  as in model (1), otherwise is proportional to  $2w^s$ .

## 2.2 Stage 2: vehicle assignment and scheduling

By our initial assumption on scheduling, any vehicle has to move end-to-end in the assigned span minimizing direction changes. This elementary idea can be implemented by different policies, for example:

- *Policy 1*: the vehicle picks up parts in the same order as it finds them in the travel, and delivers them in the same order as destinations are encountered.
- *Policy 2*: the vehicle first operates all the pick-ups in the order found in the travel; once all parts are collected, the vehicle delivers them in the order in which destinations are encountered.

Policy 1 also aims at minimizing the part stack in the vehicle, while the goal of Policy 2 is to minimize the work-in-process at the plant stations and the time each vehicle waits for another: in fact, with Policy 2, the pick-up operations precede every delivery, so we use from the very beginning the whole vehicle capacity available to free stations as far as possible. Tests conducted in [Arbib et al. 2021] demonstrated that the best scheduling method follows Policy 1. So let us next detail the way it works.

For any  $s \in S^*$ , let  $K^s$  contain the vehicles whose capacity is sufficient to accommodate all the pick-ups in  $s$ . Suppose then that span  $s$  is assigned to vehicle  $k \in K^s$ , which starts from position  $\xi_k$ . To describe the scheduling algorithm implemented for  $k$ , we partition again  $R^s$  into forward and backward requests  $F^s, B^s$  as in §2.1. If  $F^s$  is non-empty, let  $p_F$  ( $d_F$ ) denote the leftmost (rightmost) pick-up (delivery) location among its requests. Similarly, for non-empty  $B^s$ , let

$p_B$  ( $d_B$ ) denote the rightmost (leftmost) pick-up (delivery) location among its request. In example 1, one has  $p_F = 1, d_F = 14, p_B = 13, d_B = 2$ .

Assume both  $F^s$  and  $B^s$  non-empty. Then

- if  $\xi_k < a^s$  (that is, the vehicle starting point lies to the left of  $s$ ),  $k$  will first fulfill forward requests, then backward ones;
- if  $\xi_k > b^s$ ,  $k$  will first fulfill backward requests, then forward ones.

If instead  $a^s \leq \xi_k \leq b^s$ , the schedule is constructed distinguishing three cases

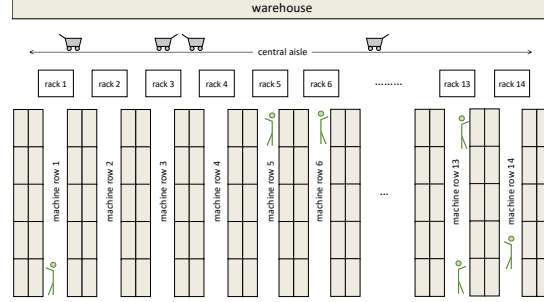
- i)  $\xi_k < d_B$ : the vehicle first fulfills the backward requests lying to its left, then changes direction and fulfills all the forward requests, then finally gets back to the remaining backward ones;
- ii)  $\xi_k > d_F$ : the vehicle first fulfills the forward requests lying to its right, then changes direction and fulfills all the backward requests, then finally gets back to the remaining forward ones;
- iii)  $d_B \leq \xi_k \leq d_F$ : we try both the above schedules and select for the vehicle the shortest one.

Finally, if  $F^s = \emptyset$ , then  $k$  has only to fulfill backward requests, so it first moves to  $p_B$  and then proceeds to fulfill all requests. A similar schedule with first pick-up in  $p_F$  is constructed if  $B^s = \emptyset$ .

Once, via the rules described above, we have collected the scheduling times  $\tau_k^s$  for each  $s \in S^*$  and  $k \in K^s$ , we match vehicles and spans (and therefore vehicles and schedules) once for all, seeking to minimize the makespan  $C_{\max}$  of all the requested services. This corresponds to solve a BOTTLENECK MATCHING PROBLEM [Garfinkel, 1971], [Burkard and Derigs, 1980]: unlike ordinary matching, where the weight of a matching is the sum of the arc weights it consists of, a matching is here weighted by the largest of its arc weights. In our case, the problem is defined on a bipartite graph with node sets  $K, S^*$ , and arcs  $(k, s)$  defined for every  $s \in S^*, k \in K^s$  and weighted by  $\tau_k^s$ . An optimal solution to this problem can be found in  $O(m^3)$  time [Pundir et al., 2015].

### 3 Industrial case study

LFoundry (<http://www.lfoundry.com>) is a company that manufactures high-tech micro-electronic components and devices in a plant located in Avezzano, Italy. The heart of the plant is the *clean room*, equipped with some 700 machines clustered according to group technology with a layout as in the scheme of Figure 1. Basically, a central aisle distributes production flows between a warehouse and a machine area, and within machine sub-areas. Machines are accessed via fourteen dedicated aisles, that lay orthogonally to the main one and are provided at their heads with a rack for in-process inventory. Parts are moved from rack to rack by a fleet of manually operated carts that commute back and forth along the main aisle, and by individual operators between the machines and the relevant head racks.



**Fig. 1.** Scheme of the clean room of LFoundry [Arbib et al. 2021].

Production is organized in small lots of identical size. Depending on product type, a lot must undergo a very large number of operations (roughly from 250 to 750). Machines can be tooled to perform specific operations, and each lot typically has the same operation type repeated many times. This, broadly speaking, gives the manufacturing system the aspect of a (very big and complex) flexible job-shop and controlling work-in-process (WIP) is one crucial issue to optimize production. Lots not currently worked by some machine, wait for operation in the racks positioned in the main aisle. Radio frequency identification devices (RFID) continuously feed the information system with the actual position of every lot waiting for or worked by a machine. These conditions suggest that a proper vehicle routing/scheduling could improve system performance and reduce the WIP.

### 3.1 Test-bed and computational settings

LFoundry provided us with very large and detailed data samples on clean room operation. Each sample describes a work shift of over 6 hours, and is formed by a series of snapshots (one per minute) of request status. Besides other information, a snapshot gives the requests pending, those in process and newly arrived ones, all recorded according to the actual practice of cart routing and scheduling. On average, 55.7% of the requests are short-distance, that is, cover  $< 6$  stations (i.e., racks) in the aisle, and only 1.8% are long-distance, that is, range over all the 14 stations. More features are given in Table 2 for the five samples used in the numerical experience.

Any two adjacent stations in the clean room are separated by 10 meters. A fleet of eight manually operated vehicles (carts) is used for transportation, each with a loading capacity of 12 lots. The (constant) cart speed is 1.2 m/s and per-lot load/unload time (15 s).

The discussion of the experimental campaign is divided into two sections:

**Table 2.** Sample work shift features.

| sample | #<br>snapshots | requests per snapshot |     |     |          | requests<br>per sample |
|--------|----------------|-----------------------|-----|-----|----------|------------------------|
|        |                | min                   | max | avg | $\sigma$ |                        |
| 1      | 374            | 38                    | 96  | 91  | 0,2      | 3.110                  |
| 2      | 351            | 32                    | 95  | 86  | 0,6      | 3.076                  |
| 3      | 343            | 41                    | 96  | 80  | 0,7      | 2.984                  |
| 4      | 342            | 45                    | 94  | 86  | 0,5      | 2.902                  |
| 5      | 347            | 54                    | 95  | 82  | 0,8      | 2.943                  |
| total  | 1.757          |                       |     |     |          | 15.015                 |

- i) In §3.2 we show the comparison of our two-stage MIP heuristic with Clarke and Wright’s heuristic (CW) on a test-bed of 50 instances. Each instance is given by a randomly selected snapshot, has between 30 and 41 requests (33.6 on average), and counts among 56 and 82 lots (70.7 on average). According to the input requirement of CW, we assume that all requests are known a priori and that carts start/end their routes at a (fictitious) depot located in a position  $h$  between the third and the fourth station.
- ii) In §3.3 we present the results achieved by our approach on the five samples described in Table 2, and compare them to current practice. To be compliant with the recorded dynamic evolution of the process, in these experiments we assume requests arrivals over time according to available data.

Algorithms were encoded in Python 3.7.6. and run on an Intel core 7,1.8-1.99 GHz processor, 16 GB RAM, operation system Windows 10 Pro 64-bit version 2004. The MIP solver adopted is Gurobi 9.0.2 with default settings using 8 threads.

### 3.2 Comparison with a Clarke and Wright heuristic

As a benchmark for assessing the MIP-based heuristic performance, we implemented a standard *cluster-first route-second* algorithm based on the classical Clarke and Wright framework [Clarke and Wright, 1964] for the VRP, integrated with a *2-opt* local search for the TSP [Croes, 1958]. This is a two-phase method in which the former step is devoted to cluster requests according to *savings*  $\sigma$  and assign clusters to carts, while the latter phase tries to minimize the route length of each cart by performing simple swaps among route components. The method pseudo-code is given in Algorithm 1 and explained hereafter.

First, the set  $C$  is populated with an initial cluster  $C_r = \{h, p_r, d_r, h\}$  per  $r \in R$ . Clusters are treated as set of positions, but a cycle  $(h, p_r, d_r, h)$  can be easily derived starting from the depot in position  $h$ , passing through  $p_r$  and  $d_r$ , then returning to the depot. Moreover,  $L_r$  is initialized to  $l_r$ .

For each pair  $C_i$  and  $C_j$  of different clusters which involve a lot volume fitting the capacity of cart  $c$ , we compute the saving  $\sigma$  on the basis of the distance  $w_{\alpha\beta} = |\alpha - \beta|$  that separates positions  $\alpha$  and  $\beta$  in a cluster. Let  $\bar{d} \in C_i$  and  $\bar{p} \in C_j$  respectively be the delivery and pickup locations with minimum distances  $w_{\bar{d}h}$  and  $w_{h\bar{p}}$  from the depot  $h$ . Then,  $\sigma = w_{\bar{d}h} + w_{h\bar{p}} - w_{\bar{d}\bar{p}}$  measures the largest potential saving one achieves by merging the cycles derived from  $C_i$  and

---

**Algorithm 1** Clarke and Wright heuristic

---

```

1: procedure CW( $I, K, R$ )
2:    $C = \emptyset$ 
3:    $C \leftarrow \text{init}(R)$ 
4:   repeat
5:      $\sigma_{\max} \leftarrow -\infty$ 
6:     for  $C_i \neq C_j \in C : L_i + L_j \leq c$  do
7:        $(\bar{d}, \bar{p}) \leftarrow \text{getPD}(C_i, C_j)$ 
8:        $\sigma \leftarrow w_{\bar{d}h} + w_{h\bar{p}} - w_{\bar{d}\bar{p}}$ 
9:       if  $\sigma_{\max} < \sigma$  then
10:         $\sigma_{\max} \leftarrow \sigma, (\iota, \kappa) \leftarrow (i, j)$ 
11:      $C \leftarrow \text{update}(C, \iota, \kappa)$ 
12:   until  $|C| \leq m$ 
13:   for  $C_i \in C$  do
14:     iter  $\leftarrow 0$ 
15:     repeat
16:        $(\tilde{C}_i, \text{iter}) \leftarrow \text{rand}(C_i, \text{iter})$ 
17:       repeat
18:          $\omega_{\max} \leftarrow -\infty, \text{flag} \leftarrow 0$ 
19:         for  $e, \bar{e} \in \tilde{E}_i : e \cap \bar{e} = \emptyset$  do
20:            $\omega \leftarrow \text{gain}(\tilde{C}_i, e, \bar{e})$ 
21:           if  $\omega_{\max} < \omega \wedge \text{feas}(\tilde{C}_i, e, \bar{e})$  then
22:              $\omega_{\max} \leftarrow \omega, (\mu, \nu) \leftarrow (e, \bar{e})$ 
23:            $(\tilde{C}_i, \text{flag}) \leftarrow \text{swap}(\tilde{C}_i, \mu, \nu)$ 
24:         until  $\neg \text{flag}$ 
25:         if  $\text{getCT}(\tilde{C}_i) < \text{getCT}(C_i)$  then
26:            $C_i \leftarrow \tilde{C}_i$ 
27:       until iter  $> N$ 
28:    $z \leftarrow \max_{C_i \in C} \{\text{getCT}(C_i)\}$ 

```

---

$C_j$ , that is, by moving from  $\bar{d}$  directly into  $\bar{p}$  without returning to (and departing again from) depot  $h$ . Among all such merging operation, we implement the one identified by the clusters indices  $\iota$  and  $\kappa$  that achieve the largest saving  $\sigma_{\max}$ :  $C$  is then properly updated by appending pickup/delivery locations of  $C_\kappa$  in  $C_\iota$  and by deleting  $C_\kappa$ . Finally,  $L_\iota$  is increased by  $L_\kappa$ .

Typically, this clustering phase is iterated while  $\sigma_{\max} \leq 0$  is found. However, as a limited number of carts is available and considering the goal of makespan minimization, we terminate this phase when no more than  $m$  clusters are gathered, trying in this way to balance the cart loads and help reduce the makespan. Notice that, in this experimental setting, assigning clusters to carts is trivial because carts are identical and share the starting position  $\xi_k = h$ ; by the way, observe also that this assumption allows us to skip the bottleneck matching step in the MIP heuristic.

Once clusters are defined, the algorithm proceeds with the routing phase. For each cluster  $C_i$  obtained, we repeat building a cycle  $\tilde{C}_i = (h, \dots, h)$  up to  $N$  times by function *rand*, which randomly sorts pickup and delivery locations in  $C_i$ . A cycle  $\tilde{C}_i$  is accepted if respects pickup and delivery precedences. We define  $\tilde{E}_i$  as the set of ordered pairs of adjacent locations  $e = (\alpha, \beta)$  in  $\tilde{C}_i$ , and  $\sum_{e \in \tilde{E}_i} w_e$  as the cycle length. For any two pairs  $e = (\alpha, \beta)$  and  $\bar{e} = (\bar{\alpha}, \bar{\beta})$  not sharing a location, function *gain* computes the cycle length reduction  $\omega$  obtained by swapping locations  $\beta$  and  $\bar{\alpha}$  in the cycle. Note that this swap reverses the path  $(\beta, \dots, \bar{\alpha})$  into  $(\bar{\alpha}, \dots, \beta)$  and brings from cycle  $(h, \dots, \alpha, \beta, \dots, \bar{\alpha}, \bar{\beta}, \dots, h)$  to  $(h, \dots, \alpha, \bar{\alpha}, \dots, \beta, \bar{\beta}, \dots, h)$ . Thus, precedence conflicts must be checked to ensure the new cycle feasibility. The feasible swap that defines  $\omega_{\max}$  is then applied to  $\tilde{C}_i$ , and new swaps are further tried until no improvement is found. The cycle underlying  $C_i$  is updated if its completion time is larger than the completion time of cycle  $\tilde{C}_i$ . Finally, the makespan  $z$  is taken as the largest completion time among all  $C_i$ .

Table 3 illustrates a comparison between the two-stage MIP heuristic and the CW heuristic with  $N = 500$ . For each of the 50 instances we report:

- number of requests  $|R|$  and total load  $\sum_{r \in R} l_r$ ;
- gap between the best known objective and the current lower bound at termination reported by Gurobi in stage 1;
- the makespan  $C_{max}$  achieved and the total CPU time in seconds, for both heuristics.

For the MIP heuristic the CPU time substantially coincides with Gurobi elapsed time, as the time required by stage 2 is always negligible. The last column reports the percentage relative gap  $\frac{C_{max}(CW) - C_{max}(MIPH)}{C_{max}(MIPH)}$ . In this experiment the time limit for Gurobi is set to 120 seconds (when it is reached, the corresponding entry reads 'tlim'): this time limit was suggested by the time allowed to make routing decision during operation.

The statistics obtained clearly show that the MIP heuristic sensibly outperforms the CW heuristic in terms of solution quality. In fact the proposed method, tailored to the rectilinear topology, always improves the makespan computed with

**Table 3.** MIP heuristic vs. CW heuristic

| $ R $ | total<br>load | MIP heuristic      |           |       | Clarke-Wright |      | MIP vs CW<br>% gap |
|-------|---------------|--------------------|-----------|-------|---------------|------|--------------------|
|       |               | Gurobi<br>opt. gap | $C_{max}$ | time  | $C_{max}$     | time |                    |
| 30    | 56            | 0                  | 295       | 6.84  | 396.67        | 0.39 | 34.46              |
| 31    | 58            | 0                  | 310       | 6.94  | 396.67        | 0.38 | 27.96              |
| 32    | 71            | 0                  | 340       | 42.73 | 415           | 0.42 | 22.06              |
| 33    | 61            | 0                  | 325       | 12.11 | 380           | 0.47 | 16.92              |
| 32    | 67            | 3.97               | 325       | tlim  | 380           | 0.44 | 16.92              |
| 31    | 65            | 1.65               | 325       | tlim  | 380           | 0.42 | 16.92              |
| 30    | 62            | 2.48               | 310       | tlim  | 396.67        | 0.38 | 27.96              |
| 33    | 74            | 0                  | 325       | 86.91 | 406.67        | 0.47 | 25.13              |
| 32    | 68            | 3.12               | 325       | tlim  | 421.67        | 0.39 | 29.74              |
| 34    | 69            | 0.78               | 340       | tlim  | 390           | 0.55 | 14.71              |
| 34    | 71            | 0                  | 340       | 74.92 | 390           | 0.44 | 14.71              |
| 32    | 73            | 3.05               | 325       | tlim  | 391.67        | 0.56 | 20.51              |
| 33    | 74            | 1.53               | 325       | tlim  | 380           | 0.48 | 16.92              |
| 33    | 73            | 5.22               | 340       | tlim  | 380           | 0.61 | 11.76              |
| 33    | 74            | 7.41               | 340       | tlim  | 380           | 0.48 | 11.76              |
| 34    | 72            | 0                  | 325       | 66.19 | 393.33        | 0.42 | 21.03              |
| 31    | 73            | 1.5                | 325       | tlim  | 373.33        | 0.41 | 14.87              |
| 32    | 73            | 0                  | 325       | 75.09 | 380           | 0.53 | 16.92              |
| 36    | 69            | 4.72               | 325       | tlim  | 373.33        | 0.52 | 14.87              |
| 35    | 66            | 3.97               | 325       | tlim  | 405           | 0.5  | 24.62              |
| 34    | 64            | 0                  | 325       | 5.08  | 408.33        | 0.48 | 25.64              |
| 36    | 63            | 4                  | 308.33    | tlim  | 408.33        | 0.5  | 32.43              |
| 41    | 71            | 4.44               | 340       | tlim  | 406.67        | 0.64 | 19.61              |
| 39    | 71            | 1.49               | 340       | tlim  | 438.33        | 0.61 | 28.92              |
| 38    | 72            | 5.84               | 336.67    | tlim  | 406.67        | 0.59 | 20.79              |
| 39    | 80            | 4.86               | 355       | tlim  | 438.33        | 0.59 | 23.47              |
| 38    | 82            | 6.16               | 355       | tlim  | 423.33        | 0.63 | 19.25              |
| 36    | 80            | 4.93               | 340       | tlim  | 421.67        | 0.47 | 24.02              |
| 34    | 75            | 0                  | 340       | 64.56 | 413.33        | 0.48 | 21.57              |
| 36    | 77            | 0                  | 340       | 51.17 | 398.33        | 0.44 | 17.16              |
| 35    | 74            | 0                  | 340       | 64.95 | 381.67        | 0.47 | 12.25              |
| 31    | 70            | 0                  | 325       | 88.14 | 381.67        | 0.41 | 17.44              |
| 31    | 69            | 0                  | 325       | 75.27 | 396.67        | 0.41 | 22.05              |
| 30    | 64            | 0                  | 310       | 11.69 | 396.67        | 0.34 | 27.96              |
| 30    | 65            | 0                  | 325       | 42.7  | 381.67        | 0.45 | 17.44              |
| 32    | 66            | 1.65               | 310       | tlim  | 405           | 0.45 | 30.65              |
| 33    | 67            | 5.65               | 325       | tlim  | 381.67        | 0.41 | 17.44              |
| 33    | 69            | 7.75               | 340       | tlim  | 405           | 0.45 | 19.12              |
| 32    | 64            | 2.44               | 310       | tlim  | 376.67        | 0.39 | 21.51              |
| 33    | 68            | 7.81               | 325       | tlim  | 396.67        | 0.47 | 22.05              |
| 33    | 71            | 4.69               | 321.67    | tlim  | 393.33        | 0.33 | 22.28              |
| 33    | 70            | 2.34               | 321.67    | tlim  | 408.33        | 0.42 | 26.94              |
| 33    | 68            | 0                  | 325       | 71.38 | 378.33        | 0.45 | 16.41              |
| 33    | 75            | 2.96               | 340       | tlim  | 365           | 0.41 | 7.35               |
| 33    | 76            | 0                  | 340       | 71.25 | 380           | 0.36 | 11.76              |
| 33    | 78            | 0                  | 336.67    | 89.88 | 380           | 0.41 | 12.87              |
| 34    | 81            | 2.16               | 355       | tlim  | 365           | 0.41 | 2.82               |
| 35    | 80            | 0                  | 336.67    | 72.66 | 396.67        | 0.47 | 17.82              |
| 36    | 78            | 4.96               | 340       | tlim  | 375           | 0.45 | 10.29              |
| 37    | 79            | 4.96               | 340       | tlim  | 375           | 0.48 | 10.29              |

CW by a meaningful amount, larger than 15% in 37 out of 50 cases and larger than 20% in 23 cases. On average, the improvement obtained is 19.60%.

Notice also that such a relevant improvement is observed even if Gurobi reached the time limit in 30 out of 50 cases. Indeed, in a preliminary experiment we investigated the sensitivity of the MIP heuristic with respect to the quality of the solution computed in stage 1. In particular, we have run the experiment with a large time limit so as to let Gurobi certify optimality (this may take up to 1 hour) and we experienced that this does not yield a significant improvement to the final  $C_{max}$ . In other words, a good performance of the two-stage heuristic is already achieved with near optimal solutions of stage 1. These turned out to be quickly computed by Gurobi primal heuristic (with default settings) at very early stages of the branch-and-cut algorithm.

Such a well-behaviour of the MIP-based heuristic supported the investigation of the potential benefits of its implementation in the LFoundry production process. This is illustrated in the next section.

### 3.3 Practical impact

This subsection follows again [Arbib et al. 2021]. It is divided in three parts: *Current practice*, *Dynamic implementation* and *Results*, and describes experiments whose aim is to illustrate the behaviour of the MIP heuristic proposed, when applied to a very large number of requests arriving over time. As observed, both to be time-compliant and to leave enough freedom to manage exceptions that may possibly occur, route operation imposes a repeated application of the method to subsets of requests of adequately small size. This implies

- the addition of release dates to the R-VRPPD, that is, time windows within which requests can be considered available for processing;
- the absence of a central cart depot, as the algorithm must now input initial cart positions anywhere in the line.

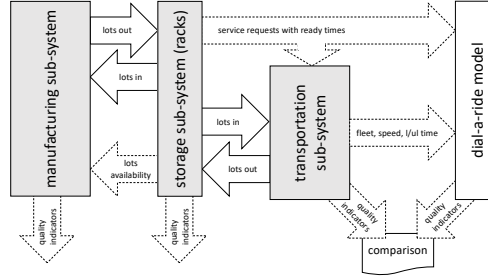
The latter issue is approached by a particular BOTTLENECK MATCHING PROBLEM, as explained at the end of §2.2. The former issue requires to deal with details not particularly demanding in terms of computation time, but still needing consideration in order to correctly concatenate subsequent problem solutions. All these details are described in the *Dynamic implementation* sub-section.

**The current practice** The current policy is based on a fixed cart routing, assessed via statistics identifying the most frequent transportation requests. Specifically, three static routes are defined: the longest, covering all the fourteen stations in the main aisle; the shortest, covering its central part; and an intermediate one, strictly contained in the former and strictly containing the latter. The cart fleet is then three-partitioned and statically assigned to those routes.

**Dynamic implementation** We have implemented our two-stage heuristic in a dynamic environment so as to evaluate its impact on the LFoundry production process. Now we have a discrete time horizon  $\mathcal{T} = 1, \dots, T$  and denote by  $R_t$  the set of requests that, at some time  $t \in \mathcal{T}$ , (i) are not yet picked up and (ii) have lots available at the respective stations. Cart  $k$  is in general available from some time  $t_k \geq t$  on, as it is supposed to conclude the deliveries assigned in the previous time lap. At that time, the cart will have a particular position  $\xi_k$  in the aisle and a particular residual capacity  $\bar{c}_k$ . As previously noticed, this means that cart initial positions generally differ from one another, so making the one-depot assumption inapplicable and requiring, on the other hand, to solve in stage 2 the BOTTLENECK MATCHING problem described at the end §2.2.

The experience was carried out with the following procedure, see also the scheme in Figure 2:

1. *Initialize.* Construct the list  $L$  of all the requests in a sample, each with the time of its arrival. Let  $t = 0$  and let  $R_0$  contain the requests of the first snapshot.
2. *Cycle.* Run the algorithm of §2 on the requests of  $R_t$ , let  $C_{max}$  and  $C_{min}$  be the time length of the longest and the shortest route of a cart. Take note of the status (positions  $\xi_k$  and minimum ready time  $t_k$ ) of all the carts at  $t + C_{max}$ .
3. *Update and Repeat.* Set  $t := t + C_{min}$ , create  $R_t$  including in it all the requests of  $L$  that arrived within  $[t - C_{min}, t]$ . Taking into account cart status, go back to step 2 and repeat until all the requests in  $L$  have been fulfilled.



**Fig. 2.** Scheme of computational experience [Arbib et al. 2021]: plant sub-systems (grey blocks), software components (white blocks), physical flows (solid arrows), main information flows (dashed arrows).

**Results** We compare the makespan determined for the requests in  $L$  with that from the current practice. The number of variables in the MIP (1)-(3) ranges

from a minimum of 2.667 to a maximum of 6.767, that of constraints from 2.669 to 6.333. The final outcome is reported in Table 4. All in all, we processed 15.015 requests for a total work time of 1.757 minutes, see column 2. The result of a numerical test based on model (1) is reported in columns 3-4, another test with the model updated by (2), (3) gives the values in columns 5-6. The former (the latter) test results in a rough 9% (30%) improvement over actual practice, amounting in absolute terms to over two hours and  $\frac{1}{2}$  (eight hours and  $\frac{3}{4}$ ) potentially saved in five work shifts.

**Table 4.** Algorithm performance. Minutes to complete operation, actual duration vs. tests with model (1) and update (2, 3), and percentage reductions.

| sample<br>id. | actual<br>duration | improvement (test) |       |               |       |
|---------------|--------------------|--------------------|-------|---------------|-------|
|               |                    | model (1)          |       | update (2, 3) |       |
| 1             | 374                | 344                | 8,1%  | 252           | 32,7% |
| 2             | 351                | 318                | 9,3%  | 254           | 27,5% |
| 3             | 343                | 308                | 10,3% | 248           | 27,7% |
| 4             | 342                | 318                | 7,0%  | 246           | 28,1% |
| 5             | 347                | 312                | 10,2% | 231           | 33,5% |
| total         | 1.757              | 1.599              | 9,0%  | 1.231         | 29,9% |

The second test gives definitely better results at a price, however, of larger CPU times. In fact, every cycle (step 2 of the above procedure) took in this case about 46,4 seconds CPU time on average, ranging from a minimum of 4,9 to a maximum of 147,8 seconds. CPU times of the first test were quite shorter: 3,6 seconds on average with a minimum of 1,4 seconds and a maximum of 6,9 seconds. We also tested our optimization method under Policy 2 of §2.2. Over all samples, this scheduling policy returned makespan of 1.477 minutes, that is 246 minutes (about 20%) worse than Policy 1 but still 280 minutes (16%) better than actual operation.

Besides makespan, it is interesting to observe how cart workloads are distributed with the proposed approaches. Table 5 gives, per sample, the mileage covered by the carts. Also under this respect Policy 1 gives the best result, returning an average cart mileage 15,7% shorter than that attained by Policy 2 (last row of Table 5). As for route length distribution, the largest cart mileage with Policy 1 is from 11,6 to 12,9 km shorter than with Policy 2. In absolute terms, Policy 1 also improves the mileage unbalance of Policy 2 of amounts ranging between 15,4% and 16,8%. However, the relative unbalance

$$U_R = \frac{\text{longest route} - \text{shortest route}}{\text{longest route}}$$

is quite large with both policies, from 48.0% to 51.6% (51.2% with Policy 2). To correctly read this information one has however to consider that program (1)-(3) tends to assign less lots to longer routes, so as to compensate travel and load/unload time.

Finally, a simple test of the robustness of the method against parameter variation was done with a new run of all samples, randomly choosing cart speed

**Table 5.** Mileage balance. Kilometres per cart to complete operation: average, maximum and unbalance (diff.) per sample and scheduling policy.

| sample id. | cart mileage (km) |      |       |          |      |       |
|------------|-------------------|------|-------|----------|------|-------|
|            | Policy 1          |      |       | Policy 2 |      |       |
|            | avg.              | max  | diff. | avg.     | max  | diff. |
| 1          | 41.4              | 63.9 | 31.5  | 49.5     | 76.7 | 37.8  |
| 2          | 42.6              | 64.2 | 30.8  | 50.6     | 77.1 | 37.0  |
| 3          | 39.3              | 62.8 | 31.7  | 46.2     | 75.4 | 38.0  |
| 4          | 39.6              | 62.1 | 31.2  | 46.5     | 74.5 | 37.4  |
| 5          | 38.5              | 63.8 | 31.5  | 48.9     | 76.6 | 37.5  |
| tot. avg.  | 41.2              |      |       | 48.9     |      |       |

and load/unload time uniformly picked in  $1, 2 \pm 0, 1$  m/s and  $15 \pm 2$  s. With both Policy 1 and 2, this experiment returned a makespan substantially close to the values found with constant parameters.

## 4 Conclusions

We have presented a cluster-first-route-second decomposition framework for a Vehicle Routing Problem with Pickups and Deliveries on a rectilinear layout. Unlike classical graph-based approaches, the clustering stage is carried out by solving a  $m$ -CENTRE-type MIP model; then, vehicles arbitrarily deployed along the line are assigned with routes by solving a BOTTLENECK MATCHING problem. An extensive computational experience has shown that this heuristic outperforms a tailored implementation of the classical Clarke and Wright heuristic. Moreover, even primal feasible solutions of the clustering MIP, quickly computed by the off-the-shelf solver, turned out to yield high quality final solutions. As such, our method looks promising for industrial implementation, which is currently under assessment by LFoundry production managers.

## References

- [Arbib et al. 2021] Arbib, C., Ranjbar, F.K., and Smriglio S. (2021) A dynamic dial-a-ride model for optimal vehicle routing in a wafer fab. *Proc. 10<sup>th</sup> Int. Conf. on Operations Research and Enterprise System*, Online Streaming, Feb. 4-6, 2021: 281-286
- [Berbeglia et al. 2007] Berbeglia, G., Cordeau, J.F., Gribkovskaia, I. and Laporte, G. (2007) Static pickup and delivery problems: aclassification scheme and survey. *TOP*, **16**: 1-31.
- [Burkard and Derigs, 1980] Burkard, R. and Derigs, U. (1980). The bottleneck matching problem. In: *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs, Lecture Notes in Economics and Mathematical Systems*, **184**: 60–71. Springer, Berlin-Heidelberg.
- [Calik et al., 2015] Calik, H., Labbé, M., and Yaman, H. (2015).  $p$ -Center Problems. In: Laporte G., Nickel S., Saldanha da Gama F. (eds) *Location Science*, Springer, Cham. [https://doi.org/10.1007/978-3-319-13111-5\\_4](https://doi.org/10.1007/978-3-319-13111-5_4)

- [Clarke and Wright, 1964] Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points *Operations Research*, **12**: 568-581.
- [Croes, 1958] Croes, G. A. (1958) A method for solving traveling salesman problems. *Operations Research*, **6**: 791-812.
- [Garfinkel, 1971] Garfinkel, R. (1971). An improved algorithm for the bottleneck assignment problem. *Operations Research*, **19**(7): 1747-1751.
- [Guan, 1998] Guan, D.J. (1998). Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, **81**: 41-57
- [Ho et al., 2018] Ho, S., Szeto, W., Huo, Y., Leung, J., Petering, M., and Tou, T. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research B*, **111**: 395-421.
- [Pundir et al., 2015] Pundir, P., Porwal, S., and Singh, B. (2015). A new algorithm for solving linear bottleneck assignment problem. *Journal of Institute of Science and Technology*, **20**(2): 101-102.
- [Ralphs et al., 2003] Ralphs, T., Kopman, L., Pulleyblank, and W., Trotter L.E. (2003) On the capacitated vehicle routing problem. *Mathematical Programming, Ser. B* **94**: 343-359. <https://doi.org/10.1007/s10107-002-0323-0>
- [Vidal et al., 2020] Vidal, T., Laporte G. and Matl, P. (2020) A concise guide to existing and emerging vehicle routing problem variants *European Journal of Operational Research*, **286/2**: 401-416.
- [Wang et al. 2006] Wang, F., Lim, A., and Xu, Z. (2006). The one-commodity pickup and delivery travelling salesman problem on a path or a tree. *Networks*, **48**(1): 24-35