A Dynamic Dial-a-ride Model for Optimal Vehicle Routing in a Wafer Fab

Claudio Arbib[1][a], Fatemeh Kafash Ranjbar[2][b] and Stefano Smriglio[2][c]

[1]*Dipartimento di Ingegneria/Scienze dell'Informazione e Matematica, and Centre of Excellence DEWS, Università degli Studi dell'Aquila, via Vetoio, L'Aquila, Italy*

[2]*Dipartimento di Ingegneria/Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila, via Vetoio, L'Aquila, Italy*

Keywords: Internal Logistics, Dial-a-ride, Optimization, Integer Linear Programming.

Abstract: We consider the problem of moving production lots within the clean room of LFoundry, an important Italian manufacturer of micro-electronic devices. The problem is modeled as dial-a-ride in a dynamic environment with the objective of makespan minimization. This general objective is achieved by minimizing, at each optimization cycle, the largest completion time among the vehicles, so as to locally balance their workloads. A cluster-first route-second heuristic is devised for on-line use and compared to the actual practice through a computational experience based on real plant data.

1 INTRODUCTION

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.

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.

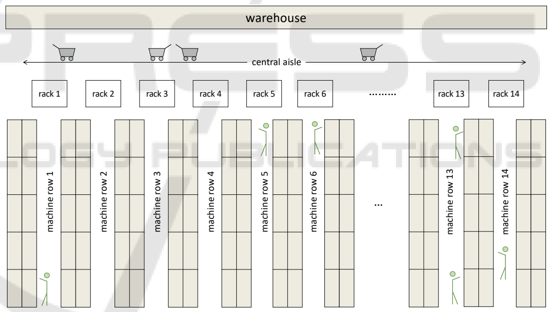

Figure 1: Scheme of the clean room of LFoundry.

Due to changes in market positioning and end-use of products, LFoundry has recently moved from mass production with large inventory levels to one oriented to individual customer needs, where smaller batches of different types are due by specified time ends. The objectives of production planning were consequently reviewed and, in this perspective, the production scheduling system is going to be completely redesigned.

Controlling work-in-process (WIP) is one among the crucial issues to be considered in order 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

[a] https://orcid.org/0000-0002-0866-3795
[b] https://orcid.org/0000-0002-4645-6818
[c] https://orcid.org/0000-0001-8152-003X

Arbib, C., Kafash Ranjbar, F. and Smriglio, S.
A Dynamic Dial-a-ride Model for Optimal Vehicle Routing in a Wafer Fab.
DOI: 10.5220/0010318402810286
In *Proceedings of the 10th International Conference on Operations Research and Enterprise Systems (ICORES 2021)*, pages 281-286
ISBN: 978-989-758-485-5

worked by a machine. These conditions suggest that a proper vehicle routing/scheduling could improve system performance and reduce the WIP.

In the past, in fact, a production with slow changes recommended to decide for a fixed cart routing, obtained via statistics identifying the most frequent transportation requests. On this basis, three static routes were 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 was then three-partitioned and statically assigned to those routes.

With the new situation, the LFoundry technical staff agreed to redesign the system so as to dynamically adapt cart routes and schedules to requests changes over time. Then, a problem arises to find optimal routes and schedules. A detailed description of the problem is given in the following §2. The solution method developed is described in §3. The outcome of a wide numerical test using real operation data is reported in §4. Finally, conclusions are drawn in §5.

2 THE PROBLEM

The problem input consists of three sets, see Figure 2:

- K set of m carts, that form the fleet used to move parts in the clean room: cart k can carry at most c lots.
- I set of n stations, corresponding to the racks headed at each machine aisle and used to temporarily host production lots.
- $R \subseteq I^2$, set of the service requests that arrive over time in a given planning horizon T: request r consists of picking up l_r lots in station p_r and delivering them to station $d_r \in I$; also, request r is associated with the time $t_r \in T$ from which the associated lots are available in p_r.

We wish to solve the following problem:

Problem 1. *Find cart routes and schedules to deliver all the lots requested, minimizing the largest completion time of any request, and also respecting possible restrictions on cart capacity and operation.*

This is a special Dial-a-ride problem, see e.g. (Ho et al., 2018), with a rectilinear layout of the vehicle stations. The problem has also the operational constraint that a cart cannot drop a lot at a station that differs from the destination, thus originating a new request to be fulfilled by other carts. Although not very common in vehicle routing problems, the simple

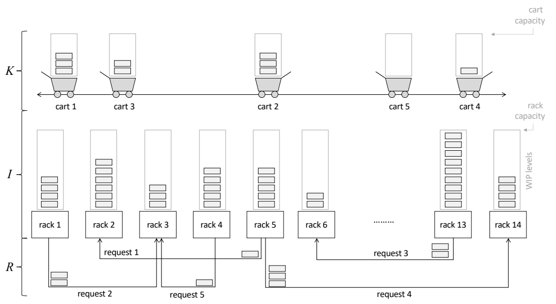

Figure 2: Problem elements.

linear layout has been investigated elsewhere for its interesting features. (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 cart with unit, or general finite, or unlimited capacity. Both studies do not capture the complexity of our case, where a fleet of carts is given, each requests involves a separate commodity, and items cannot be dropped at intermediate stations.

However, in the remainder of the paper we show how to exploit the rectilinear layout to develop a math-heuristic algorithm that, cyclically repeated in the planning horizon, solves Problem 1 in two stages.

3 A MATH-HEURISTIC

Let $R_t \subseteq R$ be the set of requests that, at some time $t \in T$, (i) are not yet picked up and (ii) have lots available at the respective stations. The linear topology of the central aisle naturally associates every subset $P \subseteq R_t$ with a minimal interval that spans between the leftmost and rightmost station involved in P, that we indicate as

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

For example, consider the requests of Figure 2, assuming $P = \{1,4,5\}$: stations $d_1 = 2, p_4 = 5, d_5 = 3$ are the leftmost involved in these requests, stations $p_1 = 5, d_4 = 14, p_5 = 4$ are the rightmost; then $a(P) = 2, b(P) = 14$.

We will in short call any such interval a *span*. There are no more spans than pairs of stations involved in R_t, i.e. no more than $\frac{q(q-1)}{2} = O(q^2)$, where $q = \min\{n, 2|R_t|\}$: in our case $n = 14$ implies 91 possible spans in the worst case. We let S denote the set of all those spans, and indicate by $w^s = b^s - a^s = b(P) - a(P)$ the width of the $s \in S$ associated with $P \subseteq R_t$. For example, the span $s = [2, 14]$ previously determined has $w^s = 14 - 2 = 12$.

Our heuristic algorithm attempts to solve Problem 1 in two stages that, according to a cluster-first route-second logic, are cyclically repeated at different time instants t:

1. Assign all the requests of R_t to m spans $s \in S$ (as many as the carts in K);
2. Given the set S^* of spans obtained from the previous stage, assign a cart $k \in K$ 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 — so that the estimated operation time (travel + load/unload) is balanced as far as possible.

In stage 2 we consider simple scheduling policies where the cart "sweeps" the assigned span from end to end, changing direction a minimum amount of times. A relevant feature of those simple policies is that a solution can easily be reconfigured at run time against possible cart capacity reduction, as well as urgent deliveries or other exceptions. Under the policy adopted, carts are scheduled with the goal of balancing operation time.

Details on the two stages are given below.

3.1 Span Selection, Request Allocation

A rough indication of the time a cart would take to cover a span s is given by

(i) the total volume l^s of requests assigned to s: large l^s correspond to long time spent in load/unload operations;

(ii) the span width w^s: the larger the w^s, the longer will supposedly be the transfer time required to cover its requests.

While w^s only depends on s, the total volume depends on the requests assigned to s. Let R_t^s denote the set of requests subsumed by s, i.e., all the r whose pick-up and delivery stations fall within the ends of s. Then:

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

where $x_r^s = 1$ if r is assigned to s and 0 otherwise. Using these variables, plus $y^s \in \{0, 1\}$ with $y^s = 1$ if and only if $s \in S$ is chosen, we can select the m spans via the following integer linear program:

$$\min_{z \in \mathbb{R}} \; z \tag{1}$$

$$\begin{aligned}
z - \frac{w^s}{v} y^s - 2u \sum_{r \in R_t^s} l_r x_r^s &\geq 0 && s \in S \\
\sum_{s \in S_r} x_r^s &= 1 && r \in R_t \\
x_r^s &\leq y^s && s \in S, r \in R_t^s \\
\sum_{s \in S} y^s &\leq m \\
\sum_{r \in R_t^s} l_r x_r^s &\leq c && s \in S \\
x_r^s, y^s \in \{0, 1\} & && s \in S, r \in R_t^s
\end{aligned}$$

where $S_r \subseteq S$ contains all the spans that can accommodate request r, v is the average cart speed and u the nominal load/unload time. Together with the objective function, the first set of inequalities defines z as the largest among the estimations of the completion time required to fulfill the requests assigned to a span. The second set of inequalities ensures that every requests is assigned to one span. The third set of inequalities activates a span when assigned a request. The remaining inequalities ensure that activated spans do not exceed available carts, and that cart capacity is in turn never exceeded.

In its first four lines, program (1) has a form close to an m-centre model, where spans corresponds to facilities, or centroids, and requests to demand points. In few words, an m-centre is a set of m points (the centroids) that minimizes the largest distance between any demand point and the closest point that belongs to that set. For a survey on m-centre problems, the reader is referred to (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 (cart capacities).

Program (1) returns:

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

One drawback of program (1) lies in the estimation of transfer time, which is assumed proportional to the span width w^s: this estimate is reasonable if the cart covers the span in one direction only, but becomes very poor if it has to go back and forth. To obtain a better estimate, we modify program (1) by distinguishing the requests in R_t^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_t^s). Precisely

$$F_t^s = \{r \in R_t^s : p_r < d_r\} \quad B_t^s = \{r \in R_t^s : p_r > d_r\}$$

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

$$\begin{array}{ccccccc} x_r^s & \leq & f^s & \leq & y^s & & r \in F_t^s \\ x_r^s & \leq & b^s & \leq & y^s & & r \in B_t^s \end{array} \quad (2)$$

for any $s \in S$. That is, span s is used if a cart 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

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

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 $2w^s$.

3.2 Cart Assignment and Schedule

Recall that by our assumption on scheduling (stage 2 of our method), unless an exception occurs, every cart moves end-to-end in the assigned span minimizing direction changes. This simple idea can be implemented by different policies, for example:

- *Policy 1*: the cart 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 cart first operates all the pick-ups in the order found in the travel; once all parts are collected, the cart delivers them in the order in which destinations are encountered.

Policy 1 also aims at minimizing the part stack in the cart, while the goal of Policy 2 is to minimize the work-in-process at the racks and the time each cart waits for another: in fact with Policy 2 the pick-up operations precede every delivery, so we use from the very beginning the whole cart capacity available to free racks as far as possible. The best scheduling method adopted in our experiments follows Policy 1 and works as described next.

In a dynamic context with requests arriving over time, 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 ξ_k in the aisle and a particular residual capacity $\bar{c}_k$. For any $s \in S^*$, let K^s contain the carts whose residual capacity is sufficient to accommodate all the pick-ups in s. Suppose then that span s is assigned to cart $k \in K^s$. To describe the scheduling algorithm implemented for k, we partition again R^s into forward and backward requests F^s, B^s as in §3.1. If F^s is non-empty, let p_F (d_F) denote the leftmost (rightmost) pick-up (drop) station among its requests. Similarly, for non-empty B^s, let p_B (d_B) denote the rightmost (leftmost) pick-up (drop) station among its request. For example, with the requests shown in Figure 2 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 cart 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 cart 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 cart 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 cart 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 τ_k^s for each $s \in S^*$ and $k \in K^s$, we match carts and spans (and therefore carts and schedules) once for all, seeking to minimize the largest completion time $C_{\max}$ 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 τ_k^s. An optimal solution to this problem can be found in $O(m^3)$ time (Pundir et al., 2015).

4 RESULTS

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 in the aisle, and only 1.8% are long-distance, that is, range over all the 14 stations. More features are given in Table 1 for the five samples used in the numerical experience.

Table 1: Sample work shift features.

	#	requests per snapshot				requests
sample	snapshots	min	max	avg	σ	per sample
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

From the data available, we deducted average cart speeds (1.2 m/s) and load/unload times (15 s); then we tested our approach and compared it to the current practice. The experience was carried out with the following procedure, see also the scheme in Figure 3:

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 §3 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 ξ_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.

We took note of the time t required to complete all the deliveries in L and compared it to the actual practice.

The algorithm was 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. In integer linear programming problems (1)-(3), the number of variables ranges from a minimum of 2.667 to a maximum of 6.767, that of con-

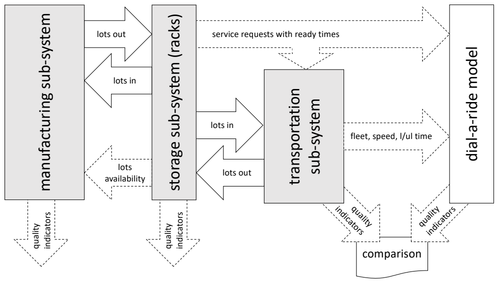

Figure 3: Scheme of computational experience: plant sub-systems (grey blocks), software components (white blocks), physical flows (solid arrows), main information flows (dashed arrows).

straints from 2.669 to 6.333. The MIP solver adopted is Gurobi 9.0.2 with default settings using 8 threads. All problems were solved to proven optimality.

The final outcome is reported in Table 2. 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 ½ (eight hours and ¾) potentially saved in five work shifts.

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

sample	actual	improvement (test)			
id.	duration	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 §3.2. Over all samples, this scheduling policy returned a completion time 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 3 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 3). 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.

Table 3: 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		

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

5 CONCLUSIONS

We tackled a challenging vehicle routing problem arising in the wafer fab of LFoundry, Italy. The problem has the form of dial-a-ride on a rectilinear topology. We developed a math-heuristic that decomposes the problem into a routing and a scheduling part. Routes are obtained by solving an integer linear program, then are assigned to vehicles by solving a bottleneck matching problem; carts schedules are found by a simple heuristic method.

The output of the method developed was used in a large scale numerical test, from which we got clear indications of possible improvement of plant operation. The improvement becomes more sensible if a more accurate estimate of scheduling time is used in the routing: this entails a trade-off between CPU time and solution quality although, in general, the method is very efficient in computational terms and compliant with on-line operation. To address the rare cases of non-compliancy, one should accelerate the resolution of program (1)-(3), which largely dominates the total solution time. Future research could then consider to exploit and strengthen, if possible, the highlighted *m*-centre structure of the model.

Further improvements of solution quality could be obtained by refining individual cart scheduling, also considering different policies. Prior to algorithm implementation in the plant, additional effort should finally be devoted to construct a simulation model of the manufacturing sub-system to observe the effect of an optimized transportation sub-system on the process of request generation.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the supportive cooperation of the team at LFoundry: Mariagrazia Colotti, Sergio D'Alberto, Fabio Di Vito, Vincenzo Greco, Massimo Ianni and Raffaele Porzio.

REFERENCES

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, 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

Garfinkel, R. (1971). An improved algorithm for the bottleneck assignment problem. *Operations Research*, 19(7): 1747–1751.

Guan, D.J. (1998). Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81: 41–57

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, 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.

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