Heuristics and Policies for Online Pickup and Delivery Problems

Martin Aleksandrov and **Pedro Barahona**
FCT, UNL, Caparica, Portugal
{m.aleksandrov, pb}@fct.unl.pt

Philip Kilby and **Toby Walsh**
NICTA, Australia
{Philip.Kilby, Toby.Walsh}@nicta.com.au

Abstract

A courier company approached NICTA to help develop a decision support tool for managing their daily pickup and delivery problem. They take orders from customers throughout the day and have to allocate one of the vehicles of their fleet to best meet the demand. This decision takes into account a number of hard (e.g. a vehicle must have capacity available for a parcel to be transported, a parcel is picked up and delivered by the same vehicle, a vehicle must wait if it arrives earlier then requested) and soft (e.g. a delivery that occurs outside a given time interval is penalized) constraints. The system we implemented to manage this Online Pickup and Delivery Problem (OPDP) selects an appropriate heuristic to decide the allocation of parcels to vehicles based on previously collected real data from the company. To verify the system working process we are proposing a novel combination of machine learning and optimisation where optimisation is used post hoc to measure the quality of these decisions. The results obtained show that the online schedules are only slightly worse than schedules produced "offline" and even outperform these for some periods of the day. We conclude by summarizing our efforts and several future improvements.

Motivation

The article is motivated by a real world problem that was presented to the Intelligent Fleet Logistics team at NICTA. Bonds Express, a local package delivery company provided some real data together with some technical information about the Sydney fleet and the human resources involved. Their request was to develop a tool assisting the company dispatchers in controlling the fleet by making multiple decisions during the day about the assignment of any new order to the vehicle which mostly satisfies particular time and other constraints.

The addressed problem belongs to the class of Online Pickup and Delivery Problems (Jaillet and Wagner 2008) with Time Windows (Mitrović-Minić 1998). Our goal is to learn dispatching policies controlling the fleet of vehicles online, rather than using an offline approach. There has been a limited amount of research on solving such problems. However, there has been some work which we report next. Cortés et al. (Cortés, Núñez, and Sáez 2008) applied a hybrid-predictive control for Dynamic PDPs, Gendreau et al. (Gendreau et al. 2006) proposed neighbourhood search heuristics for dispatching vehicles and Beham et al. (Beham et al. 2009) used a simulation-based model to realize this. However, none of them ranks heuristics and learns dispatching policies as here.

In the next section we present more detailed features of the problem and analyze their statistical properties to support a model used to complement real orders. Then the following section overviews the studied heuristics presenting some indicators of their relative merit. The system implemented is described next, which addresses the automated learning of heuristic features for specific orders in real time, as well as their subsequent selection, together with a brief discussion on the results obtained. Finally, we summarize the work done and present some future developments.

Problem and Data

The Sydney branch has 192 vehicles, having their own commodity characteristics as well as different speeds, i.e. *vehicle type* (e.g. van, motorbike, truck, etc.). In addition, they begin and finish each working day at different depots (Irnich 2000). In fact, many start and end their day at the home of the driver. Each vehicle in the company performs in one of the following three shifts: 5 a.m. - 1 p.m., 9 a.m. - 3 p.m. and 1 p.m. - 7 p.m.. The customer orders are (1) static (i.e. known in the morning of the working day) and (2) dynamic (i.e. arrive unpredictably as the day progresses). Each order has the following attributes attached to it: (1) *identifier*, (2) *number of pallets*, (3) *weight*, (4) *arrival time* (i.e. the time when the particular customer requests the demand), (5) *service time* (i.e. the time needed to load and unload the packages at the requested locations), (6) *earliest pick-up time* (i.e. the time before which the packages cannot be picked up), (7) *latest pick-up time* (i.e. the time before which the packages are preferred to be picked up), (8) *latest delivery time* (i.e. the time before which the packages are preferred to be delivered) and the (9) *locations* between which the packages are to be transported. We consider two different service times

for orders below (3 min) and above (10 min) 250 kilograms. An order that is picked up by one vehicle must be delivered by it and a driver can only be consulted about some new demands at a customer or a depot location (i.e. we do not assume vehicle diversion).

Next, we describe the real data provided by the company and the way we generated a collection of 230 datasets using some statistical regularities found in this data. Initially, the company provided information about the services performed within a single day. This list contained 1413 orders in total with 65 static, 1128 dynamic and 220 next-day demands. We consider scheduling within a single day. Each customer order in this summary is described as shown in Table 1.

id	p	w	a	s	ep	lp	ld	$\mathbf{loc}_p$	$\mathbf{loc}_d$
6829586	0	5	05:55	00:06	05:55	06:25	06:10	(−33.71,150.78)	(−34.21,150.80)
6831652	1	400	06:08	00:20	07:30	08:30	09:55	(−33.68,151.45)	(−34.29,151.46)
6831654	0	1800	06:22	00:20	07:00	09:00	11:00	(−34.14,151.28)	(−33.76,151.53)
6831655	0	1	06:46	00:06	07:00	08:00	10:00	(−33.62,151.53)	(−33.63,150.91)

Table 1: Order attributes : **id**-identifier, **p** - number of pallets, **w** - weight, **a** - arrival time, **s** - service time, **ep** - earliest pickup time, **lp** - latest pickup time, **ld** - latest delivery time and $\mathbf{loc}_p$, $\mathbf{loc}_d$ - geographical locations where the client requested services.

We studied the distributions of these features and also the regression dependencies among them. After conducting multiple statistical analyses we found two groups of independent features, i.e. (1) *weight* and *number of pallets*, and (2) *arrival*, *earliest pickup*, *latest pickup* and *latest delivery* times. However, the *weight* and the *arrival time* of the demands are predictors within these groups, respectively. In Figures 1 and 2 we show their distributions.

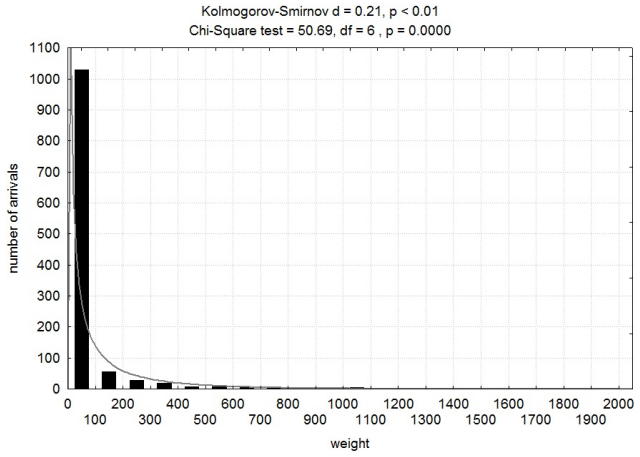

Figure 1: The distribution mass-function of the order *weight*.

The regression coefficients we obtained from the analyses are $r_{p,w} = 0.62$ between pallets and weight, and $r_{ep,a} = 0.98$, $r_{lp,ep} = 0.98$, and $r_{ld,lp} = 0.95$ between the earliest pickup and arrival times, latest and earliest pickup times, and latest delivery and pickup times.

Next, using this statistical information we generated 230 datasets. Each one of them aims to represent the information revealed in the company within a single working day. These are exploited as follows :

- Datasets 1-100 are used to assess the performance of the heuristics described in the next section. We call the tuples in these datasets *real orders* (**RO**).
- Each of the datasets 101-200 aims to simulate additional customer demands for a day from the first group and, thus, to anticipate some possible future orders. The latter are called *simulated* orders (**SO**).
- The last 30 datasets serve to evaluate the quality of the policies learnt.

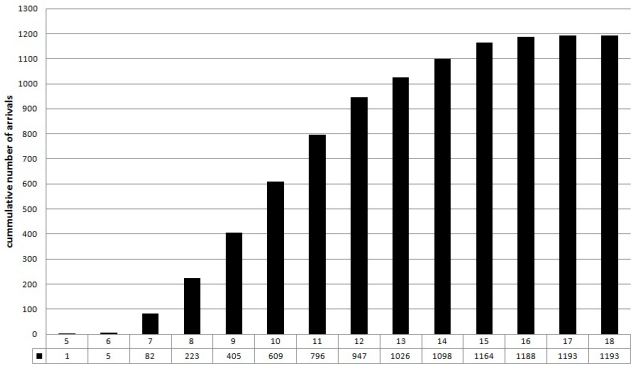

Figure 2: Cumulative function of the interarrival time.

In Table 2 we summarize some statistics regarding the generated customer errands. By m and σ are denoted the mean and the standart deviation, respectively, of the arrived orders per day. We also report the minimum ($\min$) and the maximum ($\max$) numbers of orders per day within the entire collection of datasets.

order type	m	σ	min	max	m_s	σ_s	$\min_s$	$\max_s$	m_d	σ_d	$\min_d$	$\max_d$
RO	1192	34	1074	1260	180	6	158	193	1012	29	907	1067
SO	1183	45	988	1274	1183	45	988	1274	0	0	0	0

Table 2: Total, static (*s*) and dynamic (*d*) statistics.

Heuristics

Once a customer order arrives it should be assigned to a vehicle according to some heuristic (or "rule"). Our system, presented in the next section, aims exactly at learning which rule is the most appropriate with respect to some fleet and order features. However, in this section, we firstly describe the dispatching heuristics we designed and implemented. Each of them observes the currently arrived order and the fleet parameters, and searches for the "best" vehicle according to a set of additional constraints it imposes.

Minimum Cost Minimum Cost (**MC**) heuristic calculates the cost of including a new order along each of the existing routes, meeting all hard constraints. The increase on the objective is made up of the additional travel costs for visiting the new locations, the extra expense of a possible waiting

and a certain service at those locations, any additional penalties for late services, and a supplementary cost for possible after-shift deliveries. Finally, the heuristic selects the vehicle with minimum extra expenditures.

Balanced The Balanced (**B**) rule aims to distribute the overall workload amongst the vehicles. First it computes the set of vehicles that are able to accommodate the new demand, meeting all the hard and soft constraints. If this set is empty then we consider the set of all adequate vehicles (i.e. the ones with enough free rack-space so they can accommodate the new load). For every such vehicle from this set we compute its future planned work and consider those with minimal such amount. The rule then selects the driver who can serve the new order with minimum outlay.

Current Orders Current Orders (**CO**) rule uses a Vehicle Routing solver to entirely re-route and re-schedule the curently known orders whenever a new demand arrives at the system. This reordering process consists of (1) allocating a vehicle for the new demand, (2) removing already serviced orders and (3) leave the demands whose packages have only been picked up to the same vehicles as they must be delivered by them (i.e. because packages cannot be transferred between vehicles or stored at some intermediate locations). On the contrary, if a package has not been picked up it could be serviced by another driver.

Shift Profitability The Shift Profitability (**SP**) dispatching rule aims to maximize the profit gained during the driver shift in terms of both number of serviced customers and total execution time. This is achieved by keeping the drivers busy (assigning more orders to them) for more or less the same amount of working horizon according to the current schedule. This heuristic meets the end-shift time boundary. It also attempts to assign the new requests into some waiting time windows along a vehicle route satisfying the hard constraints. Hence, we compute a penalty if a parcel is picked up or delivered late. In this way the rule evaluates the routes and chooses the one with minimum value.

Geographical Closeness The time features of the current schedule are meaningful fleet characteristics, however, the vehicle spatial distribution is also important. The Geographical Closeness (**GC**) heuristic takes into account exactly this distribution. The rule computes the subset of vehicles, nearby both locations of the new order and tries to send such a candidate, satisfying all soft constraints. The search starts with predefined areas around the new locations and if no adequate vehicle is found these areas are iteratively enlarged until some boundary is reached. In the latter case the set of all vehicles is considered and the rule recommends the driver able to handle the new demands with minimum extra costs.

Minimize Vehicles All of the previous heuristics minimize in some way the additional costs incurred whenever a new order occurs, allowing the use of as many vehicles as convenient. Hence, the possibility of activating an empty vehicle is quite high. In reality this would mean additional and undesired driver costs. Moreover, sometimes we prefer paying more and assigning the new parcels to a moving vehicle to contracting a new driver. This motivated us to design the Minimize Vehicles (**MV**) heuristic. It minimizes the number of vehicles still avoiding unwanted lateness. In other words, whenever a new order arrives the rule considers only the moving vehicles and only if none of them can accommodate that service, the rule recommends an empty vehicle.

Immediate Cost The Immediate Cost (**IC**) heuristic takes into account the time a particular order stays in the system. Its goal is to minimize this time whenever a new demand occurs. This is ensured by assigning the new demands to a route where it can be executed as soon as possible. The only constraints are the vehicle shift times as well as the hard and soft constraints at the locations of the new order. Finally, the route with minimum additional costs is returned. Note that this rule allows us to estimate the immediate reward of assigning the new order, whereas all of the remaining rules somehow minimize the aditional costs, frequently resulting in later services.

Random The last implemented dispatching rule is the Random (**R**) heuristic. It selects one of the previous heuristics in an uniform fashion and activates it whenever a new order arrives at the system.

We conclude this part with a note on the worst-case theoretical complexity of the implemented algorithms. Let us denote by f the fleet size (i.e. 192 in our case), by $n_{\max}$ the maximal number of packages currently assigned to a vehicle and by I the complexity that a Vehicle Routing solver achieves when assigns a new order to the current schedule. Normally the latter is polynomially related to f and $n_{\max}$. The Current Orders heuristic is the only rule that requires time proportional to $\Theta(I * (f * n_{\max}))$ supposing that the solver runs for at most some fixed amount of time irrespective of the problem size. The remaining heuristics can be implemented with polynomial complexity of $\Theta(f * n_{\max}^2)$.

Heuristic evaluation

In order to assess the decisions made by the heuristics we use a novel combination of machine learning and optimisation, in which a vehicle routing solver is called post hoc to measure the quality of individual decisions. Indigo is a Vehicle Routing solver that, given a problem specification, finds a schedule that is close to the optimal one. Finding the optimal schedule is a challenging optimisation problem above NP. The solver is randomized so it may return a slightly different schedule when run a second time with the same problem formulation. This solver is a complex combination of simple insert and local-search methodologies. Each schedule produced with Indigo is called a *suboptimal* schedule. It is a list of timetables (i.e. one per vehicle) of location assignments equiped with commodity information. Each such assignment is called *suboptimal* assignment. To evaluate the heuristic decisions let us have such a schedule and fix the orders that had been requested before a given demand has entered the system. Then we apply each rule over this "partial" suboptimal schedule together with the new order and compare its output with respect to 2 binary valuation criteria : (1) *precise vehicle* measure, which takes 1 whenever a decision coincides with the suboptimal vehicle assignment

and (2) *vehicle type* measure returns 1 if the recommended vehicle is of the same type as the vehicle accommodating that order in the suboptimal schedule or there is a vehicle in this schedule that is geographically near by the suboptimal assigned and has the same type as the recommended. If any of the measures produces 1 we call that heuristic decision *correct*, otherwise it is *incorrect*. We collected these recommendations for all *real* orders in datasets 1-100.

The rules were verified on top of two pooling strategies. During the day many customer demands occur and it may be beneficial to reoptimize the currently present orders in the system at some time instances. Thus, a pooling strategy provides exactly a scenario in which such updates are performed. We assume *once* pooling strategy in which we observe the entire working time horizon (5 a.m. - 7 p.m.) and we also consider *time-zones* pooling strategy that is based on the notion of *time slices* (Kilby, Prosser, and Shaw 1998). These are defined using the order arrival time distribution aiming to cut the entire daily information into smaller portions suitable for processing by the system. The time instances in this pooling strategy that we select are 5 a.m., 9 a.m., 11 a.m., 1 p.m., 3 p.m. and 7 p.m.. In both strategies we used Indigo to produce 10 suboptimal schedules at the reoptimization points, 5 of which consist of real orders only and the other 5 also containing simulated ones. We can view the intermediate schedules as *local* incomplete oracles having partial information with respect to the entire order sequence for the particular day and the final schedules as *global* complete oracles knowing all the suboptimal assignments for all the orders. Hence, the heuristic correctness can be measured with respect to the different local and global oracles. Note that the "global" comparison is important as in a final schedule the entire information is known and the most reasonable decisions could have been made. However, we believe that considering also the "local" comparisons would allow us to learn the policies of the local oracles (Riedmiller and Riedmiller 1999).

Results

In this section we report some of the results obtained in *once* and *time-zones*. Although, more statistics can be shown, here we leave many out for the sake of simplicity and concentrate on the major ones.

Time performance Table 3 shows the heuristic time performance in the cases when only real demands are present (**RO**) and there are also simulated orders (**RO+SO**).

orders	MC	B	SP	CO	GC	IC	MV	R
RO	33.29	27.69	10.73	$\geq 2 * 10^4$	12.52	9.15	27.97	18.27
RO+SO	116.44	100.85	15.33	$\geq 3 * 10^4$	27.8	15.57	91.93	54.31

Table 3: Once strategy : heuristic average time performance in milliseconds.

The numbers in the second row are higher due to the increased number of client orders (i.e. *real* and *simulated* ones), which implies relatively longer vehicle routes. The latter increases the search for the "best" candidate each rule performs and, hence, reflects on its execution CPU time.

Correctness Table 4 contains some correctness statistics obtained in *once* scenario. Recall, we have two valuation measures, two types of orders and eight dispatching rules. The results are averaged by the datasets 1-100 (i.e. ≈ 1000 dynamic orders per dataset).

	RO		**RO+SO**	
Rule	*precise*	*type*	*precise*	*type*
MC	293	674	265	703
B	6	403	2	450
SP	38	577	39	654
CO	581	874	575	910
GC	88	625	77	650
IC	75	616	43	661
MV	303	666	281	698
R	101	571	86	615

Table 4: Once strategy : global valuation statistics.

Notice that all the values are obtained with respect to Indigo-schedules. This oracle is just a complex heuristic and, consequently, the fact that a rule has low (e.g. **B**-heuristic with less than 10 hits) or high (e.g. **MV**-heuristic with more than 280 hits) value of correct decisions with respect to these schedules does not imply that it would achieve the same results with respect to another Vehicle Routing oracle. It simply means that the features considered by the rules are not nessesarily observed by Indigo when constructing schedules. Therefore, we keep all the rules in our further experiments except **CO**. It performed with the highest correct values, but has a prohibitively expensive execution time.

The use of simulated orders keeps the drivers busier and apparently the values for the *precise* (*type*) measure decrease (increase). The latter means that, when anticipating any future orders, the chance for a rule to "hit" the precise vehicle (correct vehicle type) decreases (increases).

	RO		**RO+SO**	
Rule	*precise*	*type*	*precise*	*type*
MC	347	591	273	628
B	5	388	2	399
SP	48	551	45	593
CO	615	731	597	652
GC	89	554	80	582
IC	78	581	46	616
MV	360	581	291	614
R	113	525	89	554

Table 5: Time-zones strategy : global valuation statistics.

In Table 5 we also present the results obtained during *time-zones* strategy. Dividing the time horizon into slices results in higher *precise* hits (except **B-heuristic**) than in *once*. On the contrary, the *type*-related hits decrease. A partial reason for that can be that local oracles produce schedules for smaller number of orders than the global one and Indigo takes this into account by using different types of vehicles.

System

This section presents the architecture of a system we designed that is able to select a rule given current parameters of the new order and the fleet. We first present this system in Figure 3. It works in cycles that can be describes as follows : it (1) receives the current schedule S and a new order o, (2) extracts the schedule ($F(S)$) and order ($F(o)$) features, (3) gives these as input parameters to a neural network, which uses a particular model to rank the dispatching heuristics ($OutputFeatures$) appropriately, (4) uses an ad-hoc rule selection procedure ($Policy$) based on this ranking, (5) applies the selected heuristic ($Action$) followed by an insertion ($Update$) of the new order o into the schedule S, obtaining a new schedule S'. Then the system waits until the next order when S' is considered as the current schedule. Our system halts at the end of the working day.

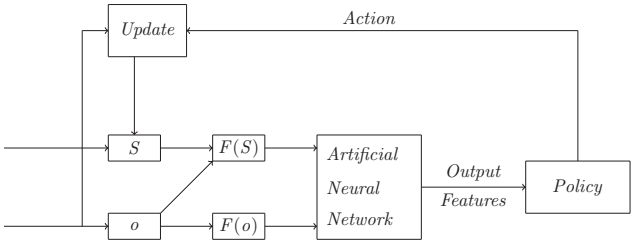

Figure 3: Recommendation system architecture.

We used cascade training to build a neural model for each pooling strategy. As we study a new problem we believe that adopting a dynamic training would better fit the problem description unless we know in advance a neural network structure that best suits these kind of problems. We also implement four dispatching policies based on the output features of the networks. Some of the results obtained are presented at the end of this section.

Neural Models

The dispatching system uses a neural network, which takes inputs describing the new order o and the current schedule S. The first group $F(o)$ consists of : (1) period of the day T ([1,4]) when o arrives, (2) number of pallets P ([1,10]) and (3) weight W ([1,10000]) describing o, (4) minimum vehicle type V ([1,14]), which is able to accommodate o and (5) the degree of dynamicity D ([1,10]), which expresses how many orders have arrived at the same moment as o has. Next, we select schedule features $F(S)$ that depend on the characteristics of o and are as follows : (6) number of active vehicles A ([1, 192]), (7) free pallet capacity FPA in the active vehicles ([0, 480]), (8) free weight capacity FWA in the active vehicles ([1, 420962]), (9) number of vehicles C ([1, 192]), which are near by o, (10) free pallet capacity FPC in these vehicles ([0, 480]) and (11) free weight capacity FWC in these vehicles ([1, 420962]). All the ranges in the brackets are the feature domains. In total we consider 11 features describing the current *state* of our problem.

Since we have two pooling strategies (i.e. *once* and *timezones*) we built two types of datasets, which are then used to build different neural models. In Table 6 we show an example of the output features the networks learn when given the following state description : $T = 1$, $P = 0$, $W = 344$, $V = 5$, $D = 1$, $A = 54$, $FPA = 93$, $FWA = 55868$, $C = 2$, $FPC = 3$, $FWC = 917$.

	Global		**Local**
Rule	*precise*	*type*	*type*
MC	0	7	6
B	0	10	6
SP	0	7	6
GC	1	2	2
IC	1	6	7
MV	2	2	0
R	0	6	6

Table 6: Output features.

Notice that the comparison was done with respect to 10 suboptimal schedules and, thus, the numbers in Table 6 vary between 0 and 10. In addition, we learn the vehicle types able to accommodate the respective order according to the schedules with and without simulated errands. Learning these separately may result in a system recommendation that sends a bigger vehicle than the suboptimal one due to the implicitly incorporated prediction that some new orders would occur in the direction of the locations of the currently new order. Finally, each dataset contains all input features, all output features and the two recommended vehicle type features.

Ad-hoc policies

After the networks are trained, they are able to rank the dispatch heuristics given a particular input parameter configuration. Then a predefined policy selects a heuristic. Let $\overline{O} = (\overline{h_p}, \overline{h_g}, \overline{h_l})$ be an output vector where $\overline{h_p}$, $\overline{h_g}$ and $\overline{h_l}$ are the value vectors of the *precise*, *type* global and *type* local heuristic features (see Table 6). Then we define the following four ad-hoc policies:

- $\pi_{PG}(\overline{O}) = \max_{h_i \in H}(\overline{h_p}(i) + \overline{h_g}(i))$
- $\pi_{pg}(\overline{O}) = \min_{h_i \in H}(\overline{h_p}(i) + \overline{h_g}(i))$
- $\pi_{PLG}(\overline{O}) = \max_{h_i \in H}(\overline{h_p}(i) + \overline{h_l}(i) + \overline{h_g}(i))$
- $\pi_{plg}(\overline{O}) = \min_{h_i \in H}(\overline{h_p}(i) + \overline{h_l}(i) + \overline{h_g}(i))$,

The first policy π_{PG} selects the rule with the maximal joint sum of the two global features. On the contrary, π_{pg} prefers the heuristic with the minimal such value. Policy π_{PLG} dispatches the new order according to the rule achieving the maximal joint sum of all the measures and the last action-mapping (i.e. π_{plg}) selects the heuristic with the minimal such score. For instance, if we consider Table 6 again, then $\pi_{PG} = \mathbf{B}$ (i.e. with sum of 10), $\pi_{pg} = \mathbf{GC}$ (3), $\pi_{PLG} = \mathbf{B}$ (16) and $\pi_{plg} = \mathbf{MV}$ (4).

Results

The system introduced ran over 30 datasets of orders ($\approx$ 30 000 orders). In what follows we present some of the results obtained in *once* and *time-zones*. As we could not compare these with either the real dispatch decisions or other online system, Table 7 and 8 show the costs obtained using Indigo and pursuing the policies PG, pg, PLG and plg.

	Indigo	PG	pg	PLG	plg
driver wages	7878.25	8226.28	7718.46	8130.49	7303.48
vehicle costs	1314.21	3631.33	3923.69	3585.26	3501.09
penalties	0.00	1342.76	9088.96	3688.95	1544.94
final	9192.46	**13200.37**	20731.11	15404.70	**12349.51**

Table 7: Policy costs in *once*.

We can observe that the first two best results are 34% (plg) and 44% (PG) worse than the "offline" costs achieved when Indigo knows all the orders in advance. These scores are also motivating as running Indigo each time a new order occurs would be a time-consuming process ($\approx$ 5 - 10 min) that could result in customer dissatisfaction and even possible service losses. As opposed to that, our system includes the new order into the current schedule in milliseconds.

In Table 8 we report the same statistics for the time slice 1 p.m. - 3 p.m.. In this scenario Indigo knows only the orders till 3 p.m. which can be a reason why the results obtained pursuing π_{pg} and π_{plg} are slightly better. Another reason for that can be that these policies select mostly rules with minimal scores (e.g. **B**). The last means that a new vehicle is decided more likely, however, Indigo does not observe this as part of its objectives and tries to use the moving vehicles instead. The cost in the rest time slices reaches up to 16.5% and 11% more than the Indigo-cost following π_{pg} and π_{plg}, and up to 24% and 27% following π_{PG} and π_{PLG}. In this pooling setting we can even decide how many drivers to contract and to release home in the different slices of the day.

	Indigo	PG	pg	PLG	plg
driver wages	7242.01	7233.16	6998.44	7233.31	7002.42
vehicle costs	1189.87	1481.55	1248.63	1493.83	1208.48
penalties	0.00	198.78	100.61	233.83	**8.83**
final	8431.88	8913.49	**8347.68**	8960.97	**8219.73**

Table 8: Policy costs in *time-zones* between 1 and 3 p.m..

Table 9 shows another set ot statistics regarding the schedules built pursuing our policies such as driver active performance, total waiting time, etc. As we can see the policy with the lowest cost achieved (i.e. plg) uses 177 vehicles, which might be inappropriate in some days, but has performed minimum late services (80). Policies PG and PLG use reasonably smaller number of vehicles (i.e. 98 and 96) to accommodate all the orders. A benefit of all action-mappings is the significant reduction of the vehicle waiting time.

Summary and future directions

We have considered an instance of an Online Pickup and Delivery Problem assuming real-time fleet and customer, time and other constraints. Taking these into account we implemented eight dispatch heuristics selecting most adequate

	Indigo	PG	pg	PLG	plg
active vehicles	68	98	178	**96**	177
requests/driver	18	13	7	13	7
waiting time[h]	76.40	**9.86**	21.54	11.18	18.37
late customers	0	209	148	380	**80**

Table 9: Policy statistics in *once*.

vehicles to accommodate the arriving orders. We generated 230 datasets, specifically adapted to our problem. We trained several neural networks for two pooling strategies and designed a system that outputs the best rule to be applied according to one of four predefined policies. Although, in the *once* policy, our online system achieved 34% worse results than Indigo, in the *time-zones* policy, it outperformed the oracle during some time slices. Based on the results obtained we are considering to improve the heuristic selection procedure by possibly defining more policies and, in addition, also by incorporating several constraints into the decision-making process, namely: (1) additional commodities (e.g. volume, length), (2) vehicle diversion, (3) cross-utilization of vehicles, (4) using real-time distances, (5) traffic congestion and (6) multiple-dispatcher fleet management.

Acknowledgments

- Martin Aleksandrov was partially supported by NICTA.
- Toby Walsh was supported by Asian Office of Aerospace Research & Development (AOARD, grant FA2386-12-1-4056).

References

Beham, A.; Kofler, M.; Wagner, S.; and Affenzeller, M. 2009. Agent-based simulation of dispatching rules in dynamic pickup and delivery problems. *LINDI* 2:1–6.

Cortés, C. E.; Núñez, A.; and Sáez, D. 2008. Hybrid adaptive predictive control for a dynamic pickup and delivery problem including traffic congestion. *IJACSP* 22(2):103–123.

Gendreau, M.; Guertin, F.; Potvin, J.; and Séguin, R. 2006. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *TR Part C* 14(3):157–174.

Irnich, S. 2000. A multi-depot pickup and delivery problem with a single hub and heterogeneous vehicles. *EJOR* 122(2):310–328.

Jaillet, P., and Wagner, M. R. 2008. Online vehicle routing problems: A survey. *OR/CSIS* 43(2):221–237.

Kilby, P.; Prosser, P.; and Shaw, P. 1998. Dynamic vrps : A study of scenarios. *APES* 53.

Mitrović-Minić, S. 1998. Pickup and delivery problem with time windows: A survey. *SFU CMPT TR* 12:669–685.

Riedmiller, S. C., and Riedmiller, M. A. 1999. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. *IJCAI* 2:764–771.