A periodic optimization approach to dynamic pickup and delivery problems with time windows

Farzaneh Karami[a,*], Wim Vancroonenburg[a,b], Greet Vanden Berghe[a]

[a]*KU Leuven, Department of Computer Science, CODeS & imec, Gebroeders De Smetstraat 1, 9000 Gent, Belgium*
[b]*Research Foundation Flanders - FWO Vlaanderen*

Abstract

In dynamic pickup and delivery problems with time windows (PDPTWs), potentially urgent request information is released over time. This gradual data availability means the decision-making process must be continuously repeated. These decisions are therefore likely to deteriorate in quality as new information becomes available. It is still believed that the state-of-the-art for this problem remains far from reaching maturity due to the distinct absence of algorithms and tools for obtaining high-quality solutions within reasonable computational runtimes. This paper proposes a periodic approach to the dynamic PDPTW based on buffering. More specifically, a two-step scheduling heuristic which consists of cheapest insertion followed by a local search. The heuristic's performance is assessed by comparing its results against those obtained by a Mixed Integer Linear Programming model which operates under the assumption that all information is available in advance. Results illustrate how the performance is impacted by urgency levels, the degree of dynamism associated with request arrivals and re-optimization frequency. The findings indicate that increases in dynamism improve solution quality, whereas increases in urgency have the opposite effect. In addition, the proposed approach's performance is only slightly affected by re-optimization frequency when changing these two characteristics.

Keywords: Buffering, Dynamic pickup and delivery problems, Dynamism, Logistics, Urgency

1. Introduction

While academic scheduling is often concerned with static problems for which all information is available beforehand, many real-world operational decision-making problems (particularly those concerning logistics) are inherently dynamic. This means that their input is revealed or changes over time. As per the definition offered by Ausiello et al. (2012), a static optimization problem $\mathcal{P}$ is represented by a quadruple (Θ, S, Obj, Opt) where Θ is the set of instances. For any $\theta \in \Theta$, $S(\theta)$ is the set of feasible solutions provided by function S. Obj is a function returning the objective value for any pair $(\theta, s) \in \Theta \times S(\theta)$, with the optimization goal being $Opt \in \{Min, Max\}$. By contrast, dynamic problems are characterized by the continuous arrival of information which modifies the input $(\theta \to \hat{\theta})$. Since θ is a part of the optimization problem, the problem itself undergoes a modification: $\mathcal{P}(\theta, S, Obj, Opt) \to \hat{\mathcal{P}}(\hat{\theta}, \hat{S}, Obj, Opt)$. To accommodate this evolution the new problem $\hat{\mathcal{P}}$ must be addressed. Therefore, any algorithm for such a dynamic problem must

[*]Corresponding author
Email address: `farzaneh.karami@cs.kuleuven.be` (Farzaneh Karami)

solve a sequence of static sub-problems as re-optimization *steps*. At each step i, θ_i is processed. Each θ_i is referred to as an *input element*. The period of time between two consecutive steps is referred to as the *execution time* and is denoted by ET. This value limits the computational time available for an algorithm to generate its solution.

Dynamic problems necessitate a mechanism to dynamic input which describes when input elements are declared over time. An input element revelation rule is a mechanism that accounts for a dynamic problem's updates during the process of input element construction for each optimization step. Thus, this rule is employed to introduce a new input element θ_i to the input sequence $\phi=(\theta_1, \theta_2, \ldots, \theta_n)$. Two examples of input element revelation rules are: (1) each θ_i is declared at its own unique release time and (2) θ_i is declared only after θ_{i-1} has been executed. Applying different input element revelation rules to the same problem will result in different input element sequences. A rule set defines a number of restrictions on solution $S(\theta_i)$ for problem $\mathcal{P}(\theta_i, S, Obj, Opt)$ (Dunke and Nickel, 2016). These rules describe what constitutes a feasible solution. As such, utilizing two different rule sets r and r' will result in two different problems: $\mathcal{P}(\theta_i, S, Obj, Opt)$ and $\hat{\mathcal{P}}(\theta_i, \hat{S}, Obj, Opt)$. A *dynamic algorithm* should therefore continuously generate input elements while simultaneously monitoring results in order to make necessary updates.

This paper considers pickup and delivery problems with time windows (PDPTWs) as a case study. These problems are ubiquitous throughout the logistics sector and involve the picking-up of items from one location and their delivery to another. Requests often require handling before a certain time. From a modeling point of view, this may be considered as the scheduling and assigning of requests to a fleet of vehicles with the objective being to minimize tardiness. In many real-world logistic situations no prior information is available regarding the number of requests, their respective locations and time windows. Such contexts correspond to the *dynamic* PDPTW which is characterized as uncertain due to incomplete prior information.

Handling such uncertainty remains a significant obstacle for researchers. The quality of optimization decisions made with respect to the data available at that particular moment in time is likely to deteriorate as new information becomes available. This inherent uncertainty implies that the global solution acquired for a given problem is not necessarily optimum, even when each individual re-optimization has been solved to optimality. There is unanimous agreement among professionals concerning the importance of accommodating unexpected events in dynamic PDPTWs (Speranza, 2018). Further difficulty is incurred by the fact that the dynamic PDPTW has been shown to be NP-hard and even more so in the case of data uncertainty (Kouki et al., 2009). Due to this, modeling, optimization and solution evaluation methods for the dynamic PDPTW are still far from reaching high-quality standards (Psaraftis et al., 2016).

While in a static context an algorithm can be assessed by its single objective value, this is not the case in a dynamic context where there are many objective values corresponding to the problem's multiple steps. Having this sequence of objective values along with the objective value at the end of scheduling horizon leads us in the direction of questions such as how well do algorithms accommodate continuous input modifications? Do frequent problem changes affect optimization accuracy? Another challenge is assessing a dynamic algorithm's performance. A valid performance measurement method must be general enough to be able to evaluate the performance of any dynamic algorithm regardless of its particular characteristics.

Three primary questions consequently arise which motivate the present paper: (i) what is an appropriate algorithm for dealing with input data uncertainty and how well does this algorithm respond to continuous modifications, (ii) how do frequent request arrivals affect optimization accuracy, and (iii) how should the algorithm's performance be evaluated?

The first research question is concerned with what constitutes an appropriate algorithm for dealing with input data uncertainty and how well this algorithm will respond to continuous modifications. To address this research question, the present paper proposes an optimization approach to the dynamic PDPTW which iteratively buffers sets of request arrivals and allocates them to available vehicles, thereby minimizing operational expenses while maintaining high quality service levels (Section 4, Figure 4). Data uncertainty associated with request arrivals is handled as much as possible by way of buffering request arrivals.

The second research question considers how the frequency of request arrivals affects optimization accuracy. To address this question, the effects of the degree of dynamism and urgency of requests as well as the re-optimization frequency upon solution quality have been explored. In Section 5.3 these local and global effects are illustrated by Figures 8-11.

The third research question is concerned with how the algorithm's performance should be evaluated. A new performance metric (Section 5.1) describes how much worse an algorithm $\mathcal{A}$ with dynamically occurring information performs compared to an optimal algorithm $\mathcal{B}$ which assumes that all information is available in advance. And while there exists no academic consensus concerning how to (fairly) evaluate a dynamic algorithm's performance, for the sake of simplicity we will assume that algorithm $\mathcal{A}$ is optimum if it can equal the performance of its static counterpart. Though this is clearly an unreasonable benchmark, it provides a tangible point of reference concerning performance.

This paper has three main contributions. The first is a buffering-strategy-based periodic optimization approach to dynamic scheduling problems. In addition to extending approaches (Máhr et al., 2010), the new algorithm takes into account request urgency levels. Second, a methodology is proposed to clarify when uncertainty, dynamism and urgency characteristics become relevant in terms of achieving optimality. Third, a broad set of experiments evaluate the influence of re-optimization frequency and other parameters on solution quality. The results demonstrate that the proposed periodic approach has a sufficiently small overhead to be used in real-world applications.

The remainder of the present paper is structured as follows. Section 2 reviews the related literature. A formal definition of the dynamic PDPTW is introduced in Section 3 along with a MILP formulation and the extension to its dynamic variant. Section 4 introduces a new approach to the dynamic PDPTW. Computational results concerning a two-step scheduling heuristic are presented in detail in Section 5. Finally, this paper ends by providing conclusions and delineating the scope and possibilities for future research.

2. Literature review

Operational research approaches to dynamic dispatching (scheduling) in logistics date back to the 1980's (Psaraftis, 1980). Despite this lengthy history, significant academic advances concerning dynamic logistics continue to be published frequently (Psaraftis et al., 2016). Three general approaches are available to accommodate the inherent uncertainty associated with the dynamic PDPTW: *rolling-horizon-based optimization*, *stochastic programming* and *robust optimization*. Rolling-horizon-based optimization accommodates uncertainty by repeatedly making decisions (re-optimization) over time. Stochastic programming relies on either predictive forecasts or knowledge concerning the probabilities of future events. The probability distributions' governing parameters are, however, seldom available in real-world situations (Ghiani et al., 2012). The computational effort of simulating future demand necessitates longer running times to achieve better results than rolling-horizon-based optimization (Ulmer et al., 2017). Finally, by contrast, robust optimization methods seek to construct a solution capable of satisfying any realization of the uncertain parameters.

Dynamic PDPTW instances are typically characterized by their urgency level and dynamism degree. Lund et al. (1996) originally defined the degree of dynamism as the ratio of on-line request arrivals to the total number of known requests. In a purely dynamic context with no off-line requests (which is the context considered by this paper) this definition of dynamism does not help to differentiate instances. The dynamism definitions provided by Lund et al. (1996) and Larsen et al. (2002) are only concerned with the number of request arrivals, ignoring their arrival times. Substantially different cases may therefore be regarded as identical. van Lon et al. (2016) treat urgency as an entirely distinct characteristic, separate from dynamism. In addition, their definition is independent from the scheduling horizon length, making it possible to compare instances of different horizons.

The majority of dynamic PDPTW optimization techniques focus on re-optimization to produce high quality solutions within strict time limitations (Secomandi and Margot, 2009; Psaraftis et al., 2016). The time interval associated with re-optimization is what distinguishes *reactive*, *periodic* and *delayed* approaches (Figure 1). Reactive approaches perform re-optimization upon each request arrival (Chang et al., 2003; Schyns, 2015), whereas periodic approaches postpone re-optimization until a certain criterion is met (Grötschel et al., 2001). Delayed approaches, meanwhile, postpone the initiation of the next re-optimization for an arbitrary amount of time. When enough time is available for the calculations of an update, reactive approaches can perform a full re-optimization. Unfortunately, in most practical applications this is not the case and therefore periodic approaches are more commonly employed. The advantage of periodic approaches in this regard is that the time available for re-optimization is known. However, it is less suitable for settings where requests are urgent. Kilby et al. (1998) proposed a periodic approach wherein the scheduling horizon is divided into fixed time slots determined based on the dynamism degree. Their approach neglects the urgency level of requests and thus incurs a significant amount of tardiness when immediate actions are required. Vancroonenburg et al. (2016) addressed the internal hospital logistics problem as a specific example of the dynamic PDPTW: a reactive approach was implemented and run on publicly available instances. Karami et al. (2018) addressed the same problem but instead developed a periodic approach. Their results indicate that when high quality service is targeted, reactive approaches outperform periodic approaches. Whereas when both operational expenses and high quality service are considered a periodic approach is the better choice.

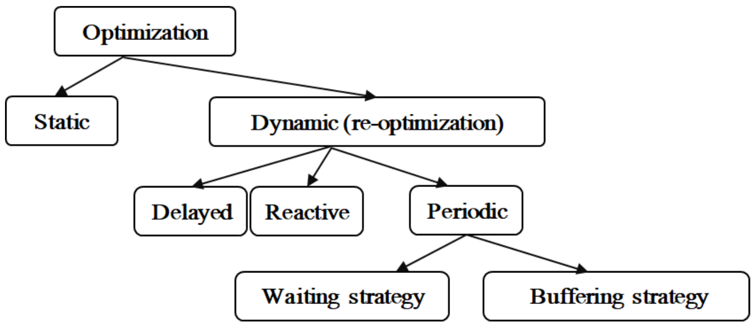
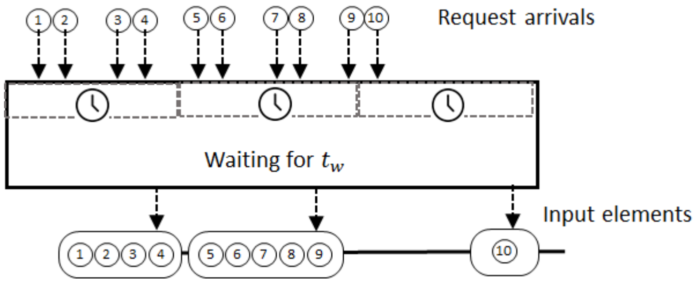
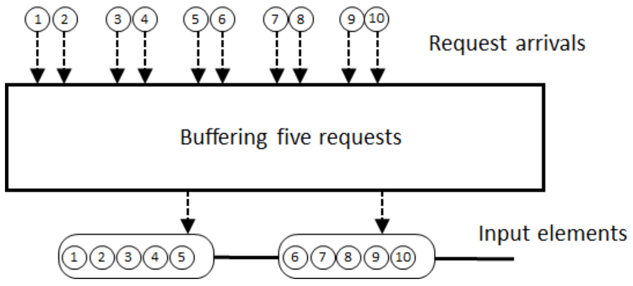


Figure 1: Optimization approaches

Two types of strategies have been proposed for improving periodic approaches: *waiting* and *buffering* (Figure 1). A waiting strategy, depicted in Figure 2(a) is concerned with vehicles and aims to minimize their travel time. However, such strategies neglect request tardiness. By contrast, buffering strategies, depicted in Figure 2(b), are concerned with requests and tend to postpone the least urgent requests until the latest possible time. Applying a buffering strategy without considering urgency and dynamism has its own drawbacks, as doing so may accumulate excessively large numbers of requests for each re-optimization. When this occurs, obtaining a high-quality solution within a reasonable time will be a difficult, if not impossible, challenge.



(a) Waiting strategy



(b) Buffering strategy

Figure 2: Two examples of the input revelation rule using a) waiting and b) buffering strategies. In both cases the scheduling algorithm may decide to reorder or redistribute the tasks, so long as the associated vehicle has not yet been dispatched.

Mitrović-Minić et al. (2004) proposed two waiting strategies: *drive-first* and *wait-first* which require vehicles to either immediately drive to or wait as long as possible before driving to their next destinations. Yang et al. (2004) and Mitrović-Minić et al. (2004) both concluded that although applying wait-first strategies to the dynamic PDPTW may decrease travel time, they will likely introduce tardiness to the solution, which is unwanted in highly urgent environments. Pureza and Laporte (2008) later combined waiting and buffering strategies into a new algorithm which facilitates the insertion of future requests while minimizing the travel time and the number of rejected requests. However, as with earlier research, their approach continues to neglect request tardiness.

Table 1 summarizes the most relevant research which has been conducted with respect to the dynamic PDPTW as well as their corresponding deterministic optimization approaches. These approaches can be sub-categorized by way of several characteristics: (1) objectives, (2) time window types: soft or hard, (3) number of depots: single or multi, (4) optimization problem: routing or scheduling, (5) data type: deterministic or stochastic, (6) uncertainty type: request arrivals, service time/travel time and vehicle availability, (7) fleet size, (8) vehicle capacity, (9) optimization approaches: offline or online, (10) type of data: real-world or randomly-generated, and (11) whether or not they permit request rejection.

Most studies focus on operational decision-making with the intent of minimizing routing costs and lateness. The costs associated with vehicle overtime are almost entirely overlooked, despite the fact that such overtime can incur significant costs for companies. Most proposed approaches are reactive and only one paper (Máhr et al., 2010) has ever considered a periodic approach with a fixed step length of 30 seconds. Máhr et al. (2010) solved the drayage problem and iteratively used a MILP model to obtain a new feasible route with newly-captured information. If no feasible solution is generated during the interval then the last feasible interval is invoked and parsed to fix assignments and rejections. This implies that, despite the arrival of new information, the solution was unable to be improved. Moreover, it may even mean that a request had to be rejected which is unacceptable in many applications. While most studies consider capacitated vehicles, Máhr et al. (2010) focused on operational decision-making with the possibility of request rejection. However, in many dynamic logistics applications with very urgent requests, rejecting a request will harm customer satisfaction and is therefore not permitted. Periodic approaches are more accommodating when it comes to responsiveness and dealing with uncertainty, however further studies concerning the effects of re-optimization frequency upon solution quality are still needed.

	Objective	Time window		Depot		Decision problem		Data type		Approach		Uncertainty type				Vehicle capacity	Request rejection	# Vehicles	Data set
		Hard	Soft	Single	Multi	Routing	Scheduling	Deterministic	Stochastic	Offline	Online (periodic reactive)	Request arrivals	Travel time Service time	Vehicle availability					
Psaraftis (1980)	↓(travel time)			×		×		×		×		×				finite		1	numerical example
Powell et al. (2000)	↓(driver cost)	–	–	×	×	×			×	–	–	×			×	finite		1 & multi	real-world
Swihart and Papastavrou (1999)	↓(service time)	–	–	×			×		×	×	×	×	×			1		1	analytical
Fleischmann et al. (2004)	↓(lateness, travel cost)		×		×	×	×				×	×	×			finite		multi	real-world
Mitrović-Minić and Laporte (2004)	↓(travel distance)	×		×		×	×	×			×	×				infinite		infinite	randomly generated (10–1000 requests)
Mitrović-Minić et al. (2004)	↓(travel distance)	×		×		×	×	×			×	×				infinite		infinite	randomly generated (10–1000 requests)
Attanasio et al. (2004)	↑ (#served requests) ↓(travel distance)	×						×				×				finite	×	multi	real-world and randomly generated
Coslovich et al. (2006)	↑ (#served requests) ↓(travel distance)	×		×		×	×	×		×	×	×					×		randomly generated (up to 50 requests)
Barceló et al. (2007)	–	×			×	×	×	×			×	×	×	×		finite	×	multi	randomly generated
Attanasio et al. (2007)	↓(average delayed services) ↑(#serviced requests)	×	×			×		×				×				infinite	×	multi	real-world and randomly generated
Cheung et al. (2008)	↓(travel time)	×		×		×		×			×	×				finite	×	multi	randomly generated
Li et al. (2009)	↓(request cancellation, rout cost)	×		×		×	×	×			×				×	infinite	×	multi	Solomon (1987)
Bock (2010)	↓(tardiness, transportation cost, service cost, vehicle cost)		×	×	×	×	×		×		×	×	×		×	finite		multi	randomly generated (124–372 requests, 38–55 vehicles)
Máhr et al. (2010)	↓(empty travel time, # rejected requests)	×		×	×	×	×			×	×	×	×			infinite	×	multi	semi real-world (65 requests)
Beaudry et al. (2010)	↓(travel time, tardiness)		×								×	×				finite		multi	real hospital data
Berbeglia et al. (2011)	–	×		×		×	×	×			×	×				finite	×	multi	modified DARP benchmark
Kergosien et al. (2011)	↓(transportation costs, tardiness)	×			×	×	×	×			×	×				finite		multi	randomly generated (130 requests)
Berbeglia et al. (2012)	↓(rout cost) ↑(# serviced requests)	×		×	×	×	×	×			×	×				finite	×	multi	real-world (Danish transporter)
Pureza and Laporte (2008)	↓(# vehicles, # rejected requests, travel distance)	×		×		×	×	×			×	×				finite	×	infinite	randomly generated (10–1000 requests)
Gan et al. (2013)	↓(average request's waiting time)		×	×		×		×			×	×				1	×	multi	randomly generated
Goel and Gruhn (2008)	↑(profit)	×			×	×		×			×	×				finite	×	multi	randomly generated
Fiegl and Pontow (2009)	↓(average flow time)	–	–		×	×	×	×				×				finite		multi	real-world
Fabri and Recht (2006)	↑(#served requests) ↓(distance traveled)	×	×		×	×	×	×			×	×				finite	×	multi	randomly generated
Ferrucci and Bock (2014)	↓(requests' lateness, travel times, service times, extra rest times and waiting times)	×		×		×			×		×	×	×			finite		multi	randomly generated
Gomes et al. (2014)	↓(operation cost) ↑(service quality)	×	×	×		×		×			×	×				finite		multi	randomly generated
Schyns (2015)	↓(distance)	×	×	×		×	×		×	×		×				finite		multi	real-world and randomlygenerated
van Lon et al. (2016)	↓(travel time, request and vehicles' tardiness)		×	×		×	×	×			×	×				infinite		multi	randomly generated
Karami et al.	↓(travel time, vehicle over time and lateness)	×	×		×	×	×	×			×	×				infinite		multi	randomly generated

Legend:
↓: minimize.
↑: maximize.
#: number of
–: not specified

Table 1: Characteristics of dynamic PDPTW studies

Any increase in storage capacity or computing power would translate into an increase concerning the problem sizes that can be handled. Gendreau et al. (2006) have investigated how the solution is affected if the number of processors is increased. As an extreme example, the memory requirement to solve the Traveling Salesman Problem (TSP) by dynamic programming grows $O(n2^n)$, while the CPU time grows $O(n^2 2^n)$ (Held and Karp, 1962). By contrast, memory requirements and CPU time evolve by only a low-power polynomial function for heuristics. This means that the greatest beneficiary when it comes to advances concerning computing power with respect to the dynamic PDPTW are heuristic approaches, at least from a practical perspective.

Various types of two-stage heuristic-based reactive approaches have been developed to solve the static PDPTW. One type attempts to reduce the number of vehicles right after obtaining the initial solution in the first stage. In the second stage, a metaheuristic is applied to decrease the travel distance. For example, Bent and Van Hentenryck (2006) use a simple simulated annealing algorithm at the first stage to decrease the number of vehicles, while the second stage involves large neighborhood search (LNS) to decrease the total travel cost. Pisinger and Røpke (2007) implemented seven different strategies for selecting requests for removal. Curtois et al. (2018) combine local search, large neighbourhood search and guided ejection search in a novel way to exploit the benefits of each method. While the local search and large neighbourhood search focus on minimising travel distance, the adaptive ejection chain seeks to reduce the number of routes.

Another type attempts to minimize lateness, travel time and overtime while the number of vehicles is fixed. For example, Gendreau et al. (2006) propose tabu search with a neighboring structure based on ejection chains. The optimization procedure is run while the environment remains static. When new requests arrive, or when a vehicle has finished its pickup or delivery task, the algorithm performs its insertion and ejection moves. Beaudry et al. (2010) insert new requests using a scheme capable of satisfying real-time requirements. Once again, the second stage consists of a tabu search which attempts to improve the current solution. Meanwhile, van Lon et al. (2016) utilized cheapest insertion in the first stage and 2-opt heuristics in the second for solving the dynamic PDPTW. Finally, a third type of two-stage heuristic-based reactive approaches focuses only on minimizing travel distance. Renaud et al. (2000) simultaneously insert each delivery and associated pickup in the first stage. The second stage is an improvement procedure that uses the x-Opt heuristic. The first stage of Mitrović-Minić and Laporte (2004) consists of a cheapest insertion procedure, while the second consists of a tabu search algorithm similar to Gendreau et al. (2006).

Most research concerning the dynamic PDPTW employs a reactive approach and ignores the tightness of inter-arrival periods. They assume the inter-arrival time to be sufficiently long. However, this assumption is likely incorrect, particularly in the case of large-scale problems. This paper is based on the periodic approach which offers more control concerning runtime. Therefore, the proposed approach not only calls for a trade-off between solution quality and response time but also calls for a trade-off between solution quality and robustness.

The same level of diversity found concerning dynamic approaches also occurs with respect to the assessment methods developed for them. The measures proposed thus far range from being entirely problem-specific to near-general. Borodin and El-Yaniv (2005) proposed *Competitive Analysis* which corresponds to the ratio of a dynamic algorithm's outcome (over a predefined time period) to the optimum static outcome. The more competitive a dynamic algorithm is, the better it will approximate the optimum solution. In principle, this method requires examining all instances of a given problem and obtaining the optimal solution for the corresponding static instances, which in most cases relies on theoretical analysis of worst case behaviour. This is however not always possible. Dunke and Nickel (2016) developed a method for analyzing the dynamic algorithm which consists of a distributional analysis of both the individual outcome of an algorithm as well as its relative outcome. Despite the extensive research carried out, the literature still lacks a comprehensive investigation concerning the effect of instance characteristics and re-optimization frequency upon performance.

3. Problem definition

A comprehensive definition of the dynamic PDPTW has already been offered by Berbeglia et al. (2010). This section therefore restricts itself to providing an overview by detailing the dynamic PDPTW's primary components and objective function. Following this, a MILP formulation and its adaptation to the dynamic PDPTW are introduced, the notation for which is presented in Appendix A.

*3.1. **The dynamic PDPTW's basic concepts***

Each pickup and delivery corresponds to two separate tasks. The tasks' *locations* and their connections are represented by an undirected graph $G(N, A, C)$. N is the set of nodes corresponding to all locations. A denotes the set of arcs connecting these nodes. Finally, $C : A \to \mathbb{R}$ denotes a travel time function, mapping each arc to its travel time. The time required to perform task i is denoted by p_i and referred to as its *service time*. The time window of each task i is expressed by way of two values: the time window's beginning and end which are denoted by st_i and dt_i, respectively. Previous studies such as by Mitrović-Minić and Laporte (2004) and Angelelli et al. (2009) assume that task time windows are not tight or that tasks may be postponed, but in many logistic applications time windows may in fact be very tight. A *request* corresponds to a transportation demand and is denoted by $r = (i, j) \in R$, with R corresponding to the set of all requests and $i, j \in T$ where T is the set of all tasks. Each request consists of a pickup task i at location $n_i \in N$ and a delivery task j at location $n_j \in N$. A semi-soft time window $[st_i, dt_i)$ is associated with pickup task i: it may not be scheduled before st_i, however it may be scheduled after dt_i. A similar time window $[st_j, dt_j)$ is also associated with delivery task j. The time at which a request r becomes known is referred to as its *arrival time* a_r. It is assumed that both the pickup and delivery tasks associated with a request are known when it arrives. The information concerning a task's pickup and delivery time windows is available upon request arrival. An *urgency window* (UW), defined as the time period from a request's arrival until the end of either its pickup or delivery time window, is associated with each request $r \in R$. Requests with shorter UW lengths correspond to those which are more urgent. Furthermore, given that the pickup UW is always shorter than or equal to the delivery UW, we define urgency based on the pickup time window.

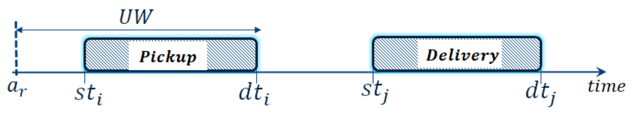


Figure 3: Pickup and delivery time windows and UW

Vehicles perform pickups and deliveries. Each vehicle k has an availability time window $[st_k, dt_k)$ during which tasks may be assigned. There is a central *depot* $\in N$ which provides the fleet of vehicles for serving the requests where all vehicles begin and end their working day. A dynamic PDPTW *instance* or *scenario* (Gendreau et al., 2006) is a triple (τ, ε, V) where τ denotes the scheduling horizon, V the fleet of vehicles, while ε consists of all request arrival events.

This paper simulates the dynamic PDPTW in a discrete event-based framework within which there are three event types: *request arrivals*, *vehicle dispatches* and *vehicle service ends*. Whenever a request arrives, a corresponding event is generated. Similarly, when a vehicle departs towards a pickup/delivery location, an event is generated to hold the vehicle's current state and destination. Finally, when a vehicle finishes a task, a service end event is generated to hold the data of the vehicle's current state and the task it has just finished.

The objective is to minimize the sum of three components which are weighted equally: total task tardiness, vehicle overtime and total travel time. Request time windows are considered semi-soft: a task may not be scheduled before the beginning of its corresponding time window, however it may be scheduled after its end. For each task i, the difference between its completion time Ct_i and its time window end is called tardiness and is calculated as follows: $l_i = Max(0, Ct_i - dt_i)$. Since all requests must be scheduled, vehicles may be assigned to requests which lie outside their availability windows. A vehicle k's overtime ot_k is the completion time of the last request performed by the vehicle minus the end of its availability time window, or 0 if negative. d_{ij} is the time it takes a vehicle to travel from location $n_i \in N$ to $n_j \in N$ along the graph's shortest path between these two locations. A vehicle's total travel time for a given day corresponds to the total time it takes to travel between its various pickup and delivery locations, in addition to its travel time to and from the depot.

3.2. ***A MILP formulation for the PDPTW***

The MILP formulation for the PDPTW is based on the formulation presented by Savelsbergh and Sol (1995). Consider the following input data:

Indices:

i/j	pickup/delivery task at a specific location.
k	vehicle.

Sets:

R	set of all requests.
T^p	set of all pickup tasks.
T^d	set of all delivery tasks.
T	set of all tasks ($T = T^p \cup T^d$).
T'	set of all tasks and a dummy task associated to the depot.
V	set of all vehicles.

Parameters and constants:

p_i	service time of task i.
$[st_i, dt_i)$	time window of task i.
$[st_k, dt_k)$	availability time window of vehicle k.

d_{ij}	travel time from location i to j.
d_{0j}	travel time from the depot to location j.
d_{i0}	travel time from the location of task i to the depot.
ω_i	tardiness weight of task i.
M	a large constant number.

The following variables are employed by the formulation:

$x_{ijk} \in \{0,1\}$	vehicle k advances to service j after performing service i.
$x_{0ik} \in \{0,1\}$	vehicle k advances to service i from its depot location.
$x_{i0k} \in \{0,1\}$	vehicle k advances to its depot after servicing task i at the end of its route.
$s_{ik} \geq 0$	the start time of servicing task i (has no meaning if i is not assigned to k).
$l_i \geq 0$	tardiness of task i.
$ot_k \geq 0$	overtime of vehicle k.

$$\textbf{Minimize} \sum_{i \in T} w_i \cdot l_i + \sum_{k \in V} ot_k + \sum_{k \in V} \sum_{(i,j) \in R} d_{ij} \cdot x_{ijk} \tag{1}$$

Subject to

$$\sum_{i \in T} x_{0ik} = 1 \qquad \forall k \in V \tag{2}$$

$$\sum_{i \in T} x_{i0k} = 1 \qquad \forall k \in V \tag{3}$$

$$\sum_{j \in T} x_{ijk} = \sum_{j \in T} x_{jik} \qquad \forall k \in V, i \in T \tag{4}$$

$$\sum_{j \in T} \sum_{k \in V} x_{ijk} = 1 \qquad \forall i \in T \tag{5}$$

$$\sum_{j' \in T'} x_{ij'k} \leq \sum_{j' \in T'} x_{jj'k} \qquad j \neq j',\ (i,j) \in R, k \in V \tag{6}$$

$$st_i \cdot \sum_{j \in T} x_{ijk} \leq s_{ik} \qquad \forall k \in V, i \in T \tag{7}$$

$$s_{ik} + p_i - dt_i \leq l_i \qquad \forall k \in V, i \in T \tag{8}$$

$$s_{ik} + p_i + d_{i0} - dt_k \leq ot_k \qquad \forall k \in V, i \in T \tag{9}$$

$$s_{ik} + p_i + d_{ij} \leq s_{jk} + M \cdot (1 - x_{ijk}) \qquad \forall k \in V, i, j \in T \tag{10}$$

$$x_{ijk},\ x_{0ik},\ x_{j0k} \in \{0,1\} \qquad \forall k \in V;\ i, j \in T \tag{11}$$

$$s_{ik}, l_i, ot_k \geq 0 \qquad \forall k \in V;\ i \in T \tag{12}$$

Constraints (2), (3) and (4) are flow constraints ensuring each vehicle must begin its route from the depot, end its route at the depot, and that after it arrives at a customer's location it must depart for another. Constraints (5) guarantee the assignment of each task. Constraints (6) ensure that if a pickup is assigned to vehicle k, then its corresponding delivery is also assigned to k. Constraints (7) enforce the commitment to the beginning of the time window. Constraints (8) and (9) derive task tardiness and vehicle overtime, respectively. Inequalities (10) are precedence constraints. They establish the relationship between a vehicle's departure time from a customer and its following delivery task. Finally, Constraints (11)-(12) define bounds for the decision variables.

This paper focuses on a dynamic PDPTW typically found in courier services with small-sized items such as letters and parcels. The fleet of vehicles is assumed to be homogeneous: all vehicles drive at a constant speed and their cargo capacity is infinite. Each instance's scheduling horizon ends when all of its pickups and deliveries have been handled. The objective function remains the same throughout the scheduling horizon. Finally, given that vehicles are rarely idle in large-scale and highly dynamic problem environments, vehicle waiting and idle times are not taken into consideration in the present paper.

*3.3. **MILP formulation adapted to the dynamic PDPTW***

A basic yet common strategy for formulating a dynamic problem is to adapt its static model such that it can respond to new information, taking into account what is currently being and has already been executed. The MILP formulation presented in Section 3.2 is therefore adapted so as to accommodate the dynamic nature of the PDPTW. Here the concept of depot must be reconsidered. At each re-optimization the vehicles' current locations will serve as their depot for the new problem. The original depot will be denoted as 'ending depot' and is common to all vehicles since they must all eventually return to it. Three new sets are also required: VK, RL and CP. VK is the set of all undispatched vehicles. RL is the set of all eligible requests, denoting all arrived requests which have neither been serviced nor scheduled. This set replaces set R used in the static context. CP is a subset of eligible requests which have been scheduled and are still in progress. A new time window constraint is also introduced to update the available time window of all currently-dispatched vehicles:

$$s_{tk} = s_{ik} + d_{ij}x_{ijk} \qquad \forall k \in V - VK, (i,j) \in CP \tag{13}$$

4. Dynamic optimization

There are many components to dynamic algorithms which distinguish them from their static counterparts. This section documents the proposed dynamic optimization algorithm and its components.

One important concept in the proposed dynamic algorithm is *request state.* At the beginning of each step t every request is in one of the following four states: *Unprocessed* (UR_t)*, Currently-processing* (CP_t)*, Currently-executing* (CE_t) *and Finished* (FR_t). Unprocessed requests are those which have arrived but which have not yet been scheduled. The set of all unprocessed requests is denoted by $UR_t = \{r \in R \mid t - ET \leq a_r \leq t,\ t - ET > 0\}$. Currently-processing requests are those which have been scheduled but which have not yet been executed. The set of all currently-processing requests is denoted by $CP_t = \{r \in R \mid a_r \leq t - ET\ \wedge\ st_r \geq t \wedge !(\textit{vehicle dispatch event}(r)),\ t - ET > 0\}$, where st_r is the beginning of the pickup time window. Currently-executing requests are those which have been scheduled and are still in progress. The set of all currently-executing requests is denoted by $CE_t = \{r \in R \mid a_r \leq t - ET\ \wedge\ st_r \geq t\ \wedge\ (\textit{vehicle dispatch event}(r)),\ t - ET > 0\}$. Finished requests are those which have been scheduled and their corresponding pickup and delivery tasks have been serviced, so they no longer require processing. The set of all finished requests is denoted by $FR_t = \{r \in R \mid dt_r \leq t\ \wedge\ (\textit{vehicle service end event}(r))\}$, where dt_r is the end of the pickup time window. At the beginning of step t the set of all requests eligible for (re)scheduling is $(UR_t \cup CP_t)$.

A visual representation of the proposed dynamic algorithm and its components is provided by way of Figure 4. This algorithm functions as a time-based periodic approach. The input element revelation rule employs a buffering-strategy and buffers request arrivals for a predefined length of time. As a rule, each vehicle must serve its next scheduled task (if one exists) immediately after completing its current task.

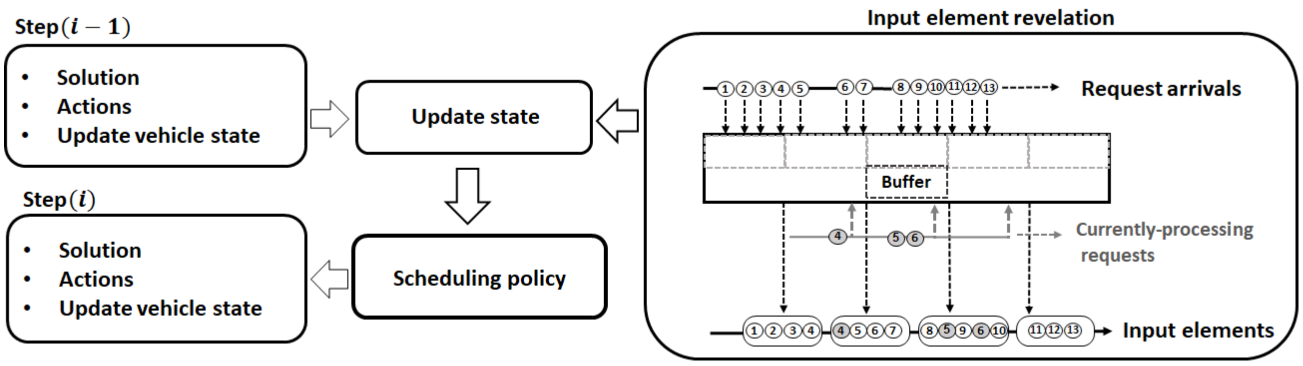


Figure 4: The proposed periodic optimization approach employing a buffering strategy.

The algorithm is executed at predefined steps $t_i^{exec} = i \times ET$ for $i = 0, 1, 2, \ldots, \lceil \frac{\tau}{ET} \rceil$. During the time-interval between two consecutive re-optimizations, any new request arrivals are accumulated and form part of the input element which will be constructed at the next step.

Two sets of time-dependent variables are associated with each θ_i: the set of its requests processing states and its set of actions. The processing states $(P_i(t) = \{(r, m) | r \in \theta_i, m \in \{UR_t, CP_t, CE_t, FR_t\}\})$ provide the state of θ_i's elements at time t and determines whether $r \in \theta_i$ is unprocessed, currently-processing, currently-executing or finished as well as the actions $(a_i(t) = \{(r, d) | r \in \theta_i, d \in \text{scheduling policy's completed decisions}\})$.

During each re-optimization step an input element is constructed for the next step. The size of the input element is governed by several factors: the total number of known requests, the decisions made by the scheduling policy during previous steps, the consequent actions taken by vehicles and the value of ET. Moreover, the input element's size is also affected by the geographic distance between pickup and delivery location pairs. For a fixed number of vehicles and requests, an increase in the service area size will result in increased average travel times for all requests. Thus, a full data investigation is required to determine an appropriate value for ET.

The concept of problem states is introduced to explain actions. The dynamic PDPTW state at time t is denoted by $state_t = (s_t^{in}, s_t^{conf}, s_t^{obj})$. The set of all input states at time t is denoted by s_t^{in}, which contains an input element and all action variables associated with its requests. At the beginning of each step t, s_t^{conf} includes all information regarding the problem's configuration and s_t^{obj} provides the objective value. State transitions triggered by events influence the algorithm input element construction. Any services associated with either currently-executing or finished s_t^{in} requests when transitioning from input state s_t^{in} to s_{t+ET}^{in} are referred to as the state's actions.

Algorithm 1 provides the pseudo-code of the proposed dynamic algorithm. This algorithm divides the entire scheduling horizon into a number of steps of length ET. Based on previous decisions and actions, a request queue consisting of all eligible input elements is constructed at the beginning of each step $t := i \times ET$ for $i = 0, 1, 2, \ldots, \lceil \frac{\tau}{ET} \rceil$. A set of rules is then applied which stipulates the conditions under which the scheduling policy must make decisions. Finally the scheduling policy is executed on the constructed request queue and its objective value and actions are stored at each step. In parallel with the scheduling policy's execution, any new request arrivals are buffered and will eventually be used to construct the input element for the next re-optimization step. The re-optimization and schedule distribution are illustrated by Figure 5.

Vehicle drivers are only informed of their next assignment upon delivery of their current request and only if the next departure is due. As a result, drivers never have full knowledge of their route. This enables the optimization procedure to update the routes, if necessary, until the latest possible moment in time.

Algorithm 1: The dynamic algorithm

```
Require: $\tau$
Require: $V$
Require: $ET$
Require: scheduling policy
1  $t \leftarrow 0$
2  state.initialize()
3  while $t \leq \tau$
4      R ← request arrivals                                  ▷ read new request arrivals (if any)
5      UR ← unserviced requests                              ▷ collect unserviced requests from past (if any)
6      INPUTS = $(R \cup UR)$
7      state ←schedule-policy.process(state, INPUTS)
8      $t \leftarrow t + ET$
9  end while
10 return
```

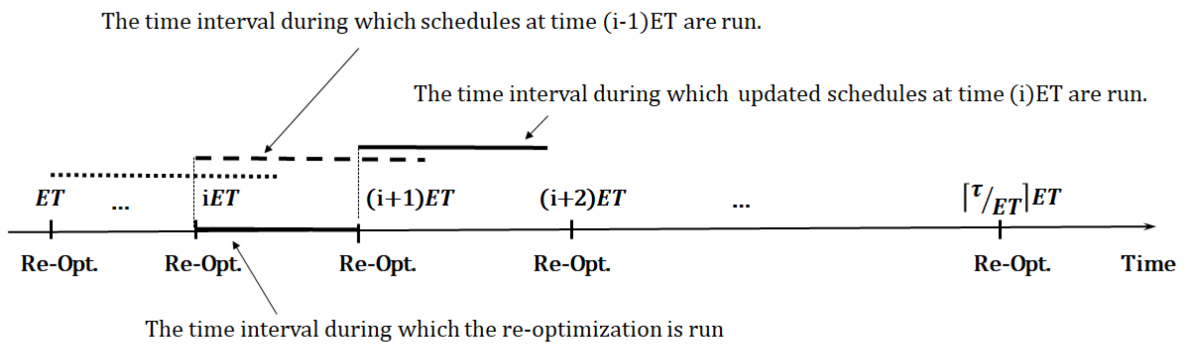

The time interval during which schedules at time (i-1)ET are run.
The time interval during which updated schedules at time (i)ET are run.
ET
...
iET
(i+1)ET
(i+2)ET
...
⌈τ/ET⌉ET
Re-Opt.
Re-Opt.
Re-Opt.
Re-Opt.
Re-Opt.
Time
The time interval during which the re-optimization is run

Figure 5: The re-optimization and schedule distribution.

The results of the dynamic algorithm over the entire scheduling horizon can be formally represented as a triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ where $\mathcal{R}$ is the set of all requests, $\mathcal{A}$ is the set of all actions and $\mathcal{C} = \{c_n | n \in \mathbb{N},\ n \leq \lceil \frac{\tau}{ET} \rceil\}$ is the set of objective values with $c_i : \theta_i \times \mathcal{A}_i \to \mathbb{R} \cup \{\infty\}$ where $\mathcal{A}_i$ denotes the set of actions taken at step i. The order in which requests appear in the sequence of actions may differ from how they were originally ordered in the input element. The processing of an input element is affected by the decisions made and actions taken during previous steps which causes the input elements' requests to transition between UR_t, CP_t, CE_t and FR_t at each t.

4.1. *Scheduling heuristic*

The two-step scheduling heuristic employs the following components.

Cheapest insertion (CI). As seen in the pseudo-code presented in Algorithm 2, at each re-optimization step CI inserts eligible requests to the available vehicles based on the current state, with the goal being to minimize the negative impact upon objective value. After determining a feasible insertion position in a vehicle's list of scheduled tasks, the start and end times of all tasks from this list are updated.

Algorithm 2: CI algorithm

```
Require: $V$
Require: $RL$
 1: for $r \in RL$ do
 2:     $a \leftarrow null$                                            ▷ a list which is initially empty
 3:     for $vehicle \in V$ do
 4:         $a.addItem(cost(insert(r, vehicle)))$                      ▷ calculate the insertion cost
 5:     end for
 6:     $c \leftarrow pop(sort(a, DESC))$                              ▷ find the lowest cost value
 7:     $vl \leftarrow$ the vehicle corresponding to $c$               ▷ select the vehicle with the lowest insertion cost
 8:     insert $r$ into the scheduled tasks$'$ list of $vl$
 9:     update                                                         ▷ update the schedule table
10: end for
11: return
```

Local search (LS). The pseudo-code, detailed in Algorithm 3, corresponds to the iterative application of the CI. At each step eligible requests are inserted into the schedule. LS's moves (ejection, move and reinsertion) are afterwards applied to improve the schedule. At each LS iteration, an eligible request is randomly selected and ejected from its corresponding vehicle's schedule after which the start and end times of scheduled requests are updated. The CI policy subsequently inserts this request into a vehicle's schedule (this could be the same or a different vehicle). The start and end times of the scheduled tasks assigned to these vehicles are updated again. A new schedule is accepted only if its objective value is better than or equal to the current schedule's objective value. This search continues until a predefined maximum number of non-improving iterations is reached.

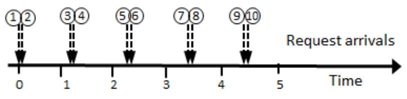
This paper assumes that ET is long enough to enable the search process to complete. Therefore, the algorithm must stop either before the next step or after a maximum number of non-improving iterations is reached. Additionally, the earliest scheduled departure time of a vehicle must occur after $ET + t$, where t is the preceding step's start time. Earlier scheduled departure times would imply that vehicles know they must depart to a task location before the algorithm has finished its calculations, which is clearly impossible.

Algorithm 3: LS algorithm

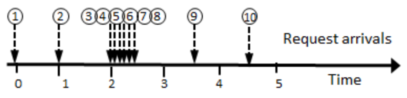
```
Require: $RL$
Require: $N_{max}$                                  ▷ maximum number of non-improving moves
Require: $V$
Require: $ET$
 1: $n \leftarrow 0$
 2: $currentCost \leftarrow cost\ of\ the\ entire\ schedule$
 3: while $n < N_{max}$ and $elapsed\ time < ET$ do
 4:     $select\ a\ random\ request\ r \in RL$
 5:     $q \leftarrow vehicle(r)$
 6:     $q.schedule.remove(r)$                      ▷ remove $r$ from $q$'s schedule
 7:     $q.schedule.update()$                       ▷ update $q$'s schedule
 8:     $newVehicle \leftarrow cheapestInsertion(r, V)$
 9:     $newCost \leftarrow cost(r, newVehicle)$    ▷ update cost by considering request $r$ is assigned to $newVehicle$
10:     if $newCost < currentCost$ then
11:         $insert(r, newVehicle)$                 ▷ add $r$ to the $newVehicle$'s schedule
12:         $n \leftarrow 0$
13:         $currentCost \leftarrow newCost$
14:     else
15:         $insert(r, q)$
16:         $n \leftarrow n + 1$
17:     end if
18: end while
19: $return$
```

4.2. ***Dynamism calculation***

Both a definition of dynamism and an illustrative example have been provided in detail by van Lon et al. (2016) who state that dynamism corresponds to the continuity of request arrivals. For example, to generate instances with different degrees of dynamism, consider $\triangle := \{\delta_0, \delta_1, \ldots, \delta_{|\varepsilon|-2}\} = \{a_{rj} - a_{ri} | j = i + 1 \wedge \forall r_i, r_j \in \varepsilon\}$, which represents the sequence of inter-arrival times for requests, where $| \triangle | := |\varepsilon| - 1$. One possible example sequence is $\triangle_a = \{0.1, 1, 0.1, 1, 0.1, 1, 0.1, 1, 0.1\}$ which includes five small bursts with intervals of 0.1 unit and four of 1 unit, as shown in Figure 6(a). By changing this order, another possible sequence for request inter-arrival times is $\triangle_b = \{1, 1, 0.1, 0.1, 0.1, 0.1, 0.1, 1, 1\}$, which has one large burst of request arrivals at the middle and two individual request arrivals before and after this burst as shown in Figure 6(b). Given the details provided by van Lon et al. (2016) Figure 6(a) has a dynamism degree of 55.5%, while Figure 6(b) has a dynamism degree of 35.1%.



(a)



(b)

Figure 6: Two examples of request arrival events with five inter-arrival times of 0.1 unit and four of 1 unit.

4.3. ***The significance of*** *ET*

Since parameter ET affects the number of requests taken as input element at each step, a large ET value may postpone some requests for a significant amount of time, which may consequently contribute towards additional tardiness. Thus, ET should be set in accordance with the urgency of request arrivals. Assume the urgency window of the most urgent request is denoted by UW_{min} and the value of ET is set to a value less than or equal to UW_{min}. As a result, there are no requests with UW smaller than ET in the requests list. Ideally ET should be set equal to UW_{min} because any smaller length represents a waste of computational resources. In most dynamic applications, UW_{min} is unknown in advance. Therefore, some investigation is required to isolate an appropriate value for ET. Figure 7 illustrates the pickup time window's relationship with ET and UW.

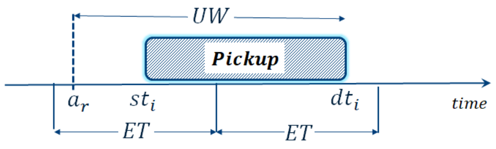


Figure 7: A visualization of ET and UW

Dynamism and urgency are two concepts which affect the dynamic algorithm in different ways. Therefore, in instances of the same scale, ET should be longer in those with less urgent request arrivals but similar degrees of dynamism. Table 2 details all four possible combinations of the dynamic PDPTW's instances' two characteristics. It can be easily seen that for the same degree of dynamism but different urgency levels: $ET_3 \leq ET_1$ and $ET_4 \leq ET_2$. However, for the same urgency level but different degrees of dynamism it is more difficult to arrive at a conclusion. Very urgent instances with low dynamism appear the hardest case for an algorithm to solve given how they often involve bursts of very urgent requests with the vehicles' number insufficient to satisfy them all. In such cases, tardiness is likely inevitable due to resource shortfalls.

	Low dynamism	High dynamism
Low urgency	ET_1	ET_2
High urgency	ET_3	ET_4

Table 2: Four possible combinations of urgency levels and dynamism degrees.

5. Computational study

A computational study is performed to evaluate the proposed dynamic algorithm. Its decisions at each step provide local perspectives while the decisions/solution over the entire scheduling horizon provides the global perspective. This study will compare the dynamic algorithm's objective values against those of the optimum static solution obtained by the MILP.

*5.1. **Performance evaluation criteria***

This section proposes two performance metrics: *local* and *global dynamic* gaps.

Local gap. This value denotes the difference between the LS scheduling policy's solution and the optimum static solution for the same input. Local gaps are calculated for each step i by $LG_i = 100 \times (\frac{Obj_{LS}(\theta_i) - OPT(\theta_i)}{Obj_{LS}(\theta_i)})$ where $OPT(\theta_i)$ is the optimum objective function value for request queue θ_i, with $Obj_{LS}(\theta_i)$ denoting the objective value of step i obtained by LS.

Global dynamic gap (GDG). This value is the difference between the objective value incurred by the dynamic algorithm over the entire scheduling horizon against the objective obtained by the MILP over the same period. The global dynamic gap is therefore calculated for the entire scheduling horizon by GDG$= 100 \times (\frac{Alg(\theta') - OPT(\theta')}{Alg(\theta')})$ where $OPT(\theta')$ denotes the optimum static solution's the objective value for request set θ', while $Alg(\theta')$ denotes the objective of the solution obtained when applying LS over the entire scheduling horizon.

The MILP solver has access to all information known at the time of re-optimization. In both the local and global cases, the MILP formulation for all instances has been solved to optimality. These results are employed to evaluate the two-stage scheduling heuristic.

*5.2. **Datasets***

This computational study employs a dataset created by the instance generator of van Lon et al. (2016). This dataset consists of instances whose characteristics exhibit a variety of dynamism and urgency configurations. The generator was employed to produce instances with eight different degrees of dynamism (20%, 30%, 40%, 50%, 70%, 80%, 90% and 100%) and five different levels of urgency (UW = 5, 15, 25, 35 and 45 minutes). Five instances were produced for each of the 40 possible combinations, resulting in a total of 200 instances. Pickup/delivery task time windows were randomly generated while respecting the constraints concerning different urgency levels and arrival times. Each instance has a size of $10km \times 10km$ with 180 requests per instance (18-30 requests per hour on average) and includes 10 vehicles with an assumed speed of 50 km/h. The depot is located in the center of this $10km \times 10km$ square. At the start of the scheduling horizon all vehicles are located at the depot. The service time of each task is 5 minutes and the entire scheduling horizon's length is 600 minutes.

*5.3. **Computational results***

Computational experiments were performed on an Intel Xeon, E5-2640, 2.60GHz processor with 16GB of RAM running Linux Ubuntu 16.04.3. The MILP formulation was solved by Gurobi 7 employing formulation (1)-(12). The scheduling policy's runtime was limited to ET. The proposed dynamic algorithm was coded in Java 1.8.

In order to evaluate the LS, its local and global performances were assessed by calculating LG_is at each step in addition to the GDG. To assess the local performance at each step, first the LS was executed on the constructed request queue and the resulting objective value was stored in addition to all actions taken. The MILP formulation was executed on the same request queue and its results were also stored. To assess global performance, another experiment was performed for each instance utilizing the MILP formulation. All these experiments were conducted with the same ET value. Finally, to study how ET affects LS global dynamic objective values and the GDG, experiments were executed wherein ET was consecutively decreased by 10 min while $ET \geq 5$. The computational runtime of the Gurobi solver is 1h, which is far longer than the time limit for the local search heuristic at each re-optimization step.

Local gap. Figure 8 indicates the local solution gap obtained by LS for instances with different urgency levels (UW lengths), wherein each box aggregates local solutions for 40 instances where the urgency level is identical but all eight different values for dynamism occur. The right boxplot presents the local solution gap for three dynamism degrees (low-20%, medium-50% and high-90%), with each box aggregating the local solutions for 25 instances where the dynamism degree is identical but all five different urgency levels occur.

Figure 8(a) shows that, on average, better local solutions can be found for instances characterized as 'highly urgent'. This is expected due to the small number of requests in the eligible request list, which makes LS more effective. It also reveals how the largest local gap decreases when either dynamism increases or urgency decreases. The small increase in the average value of the local gap (2%) when transitioning from urgency level 35 to 45 may be explained by the increased total number of eligible requests at each step. It would require longer runtimes to achieve tighter gaps.

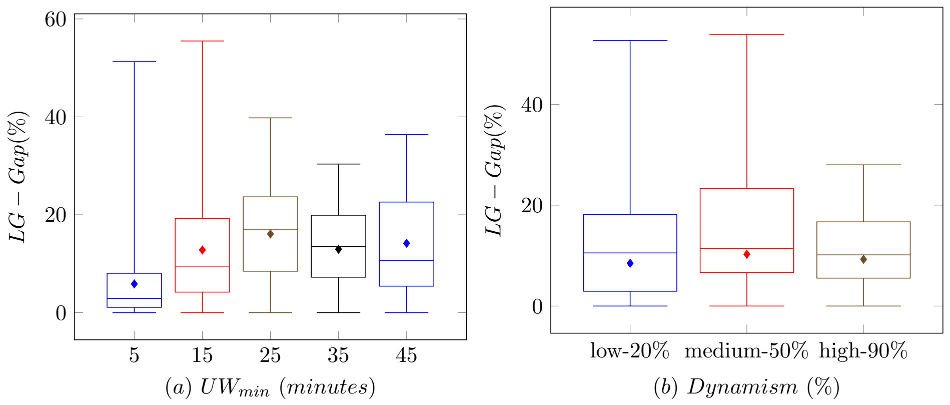
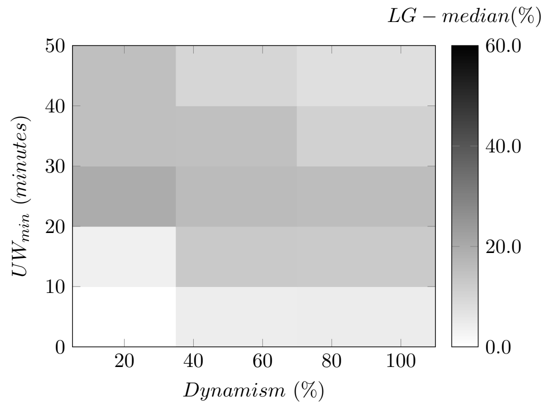


Figure 8: Local gaps (LGs) for instances with different urgency levels (UW length) 5, 15, 25, 35 and 45 minutes and three degrees of dynamism low-20%, medium-50% and high-90%. In the left boxplot, each box aggregates LGs for instances with three different dynamism degrees and the same urgency level. The right boxplot presents the LGs for instances with five different urgency levels and the same dynamism degree. The average gap for each boxplot is provided by '♦.'

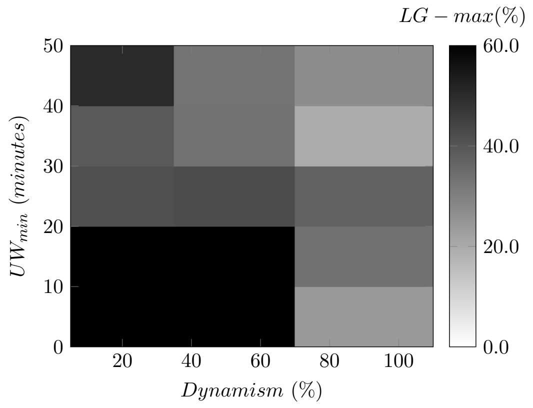
When it comes to the quality of the local solutions, a higher average LG% is expected as dynamism increases. However, the computational results do not fully support this expectation.

Instead, they demonstrate a non-linear effect, as illustrated in Figure 8(b). This is evidenced by the lack of a peak in the top right as dynamism increases. In very urgent scenarios with low degrees of dynamism, the periodic re-optimization runs in shorter steps. Therefore, the relatively small effect of high degrees of dynamism on the local solution's quality LG% can be explained by the small number of requests per buffer. Such cases often appear to be easy for the algorithms to solve within the time limit.

Meanwhile, a higher average LG% is expected as urgency increases. Again however, the computational results do not fully support this prediction, as evidenced by the lack of a peak in the top left of Figure 8(a). This effect is due to the existence of scenarios which incorporate queues smaller than the fleet of available vehicles, which means that requests are serviced quickly. Non-urgent scenarios with any dynamism degree seem to be the most difficult scenarios for the algorithms to solve. This is possibly due to request bursts associated with these scenarios. The number of vehicles demanded by these small bursts can exceed the available fleet size, which will result in tardiness.



(a)

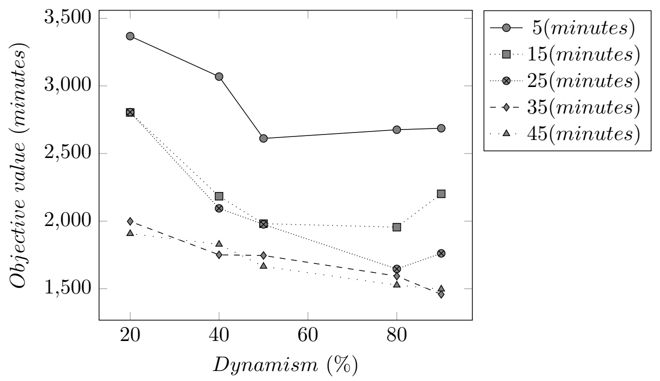


(b)

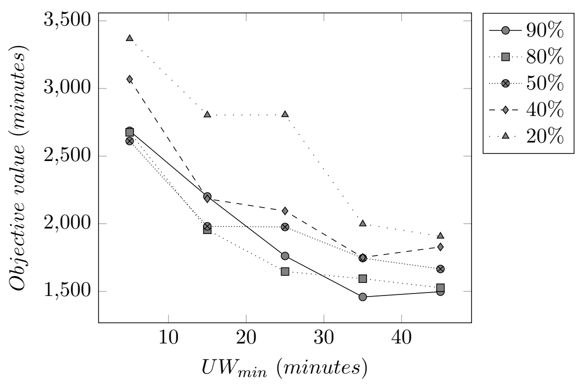
Figure 9: Median and maximum LG values for different combinations of dynamism and urgency.

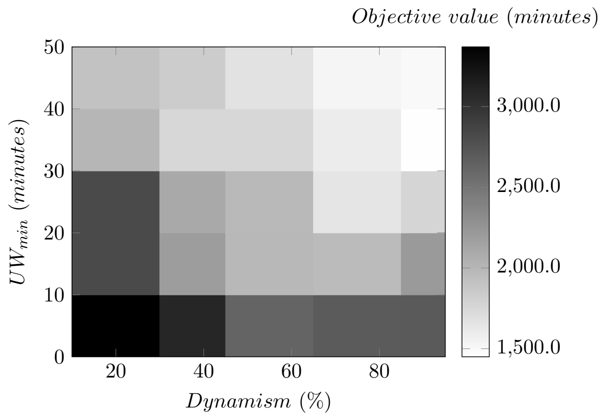
Figures 9(a) and 9(b) provide the median and maximum LG for instances with different combinations of dynamism and urgency. Figure 9(a) illustrates the homogeneous behaviour of LS in more detail, while Figure 9(b) represents the worse local behaviour of instances with different dynamism degrees and urgency levels.

Global solutions. The dynamic algorithm is analyzed in order to determine whether or not its objective function value and GDG depend on the degree of dynamism and the urgency level of incoming requests.



(a)





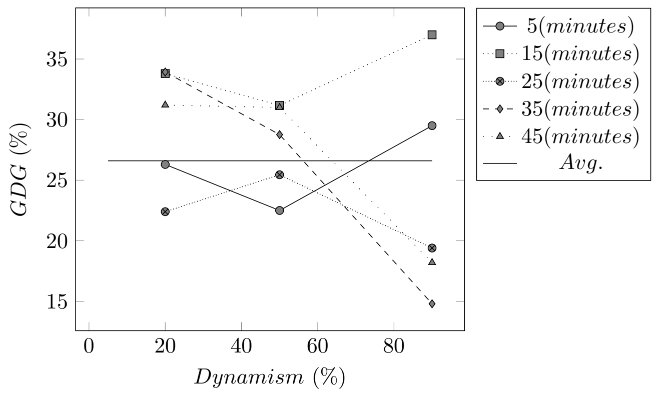
(b)

(c)

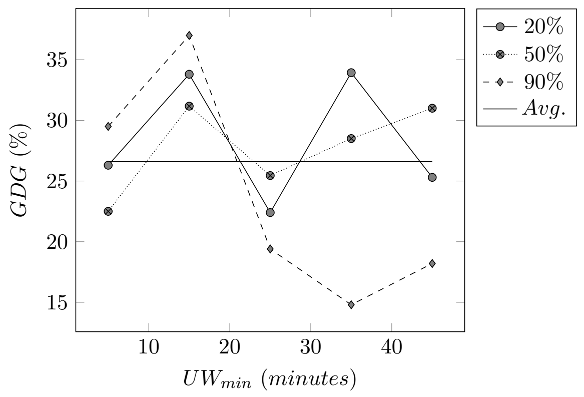
Figure 10: Objective value obtained by the dynamic algorithm. Figure (a) illustrates that the objective value will improve by increasing the degree of dynamism. Figure (b) demonstrates that the objective value will improve by increasing the *UW* length. Figure (c) illustrates how often instances of varying degrees of dynamism and urgency levels are the most/least costly.

Figures 10 provides the objective values obtained in the experiments. Figure 10(a) illustrates how for any level of urgency (*UW* length) the objective value will improve whenever the degree of dynamism increases. Figure 10(b) demonstrates how the objective value will decrease when the urgency of request arrivals decreases for any dynamism degree. Finally, Figure 10(c) illustrates how instances with low dynamism degrees and high levels of urgency are the most challenging and costly for the proposed dynamic algorithm to solve. When some requests in the eligible request queue must be handled urgently, it is natural that some delays are introduced. Furthermore, when requests are highly urgent there is less possibility for minimizing the travel time which eventually results in a worse objective value.

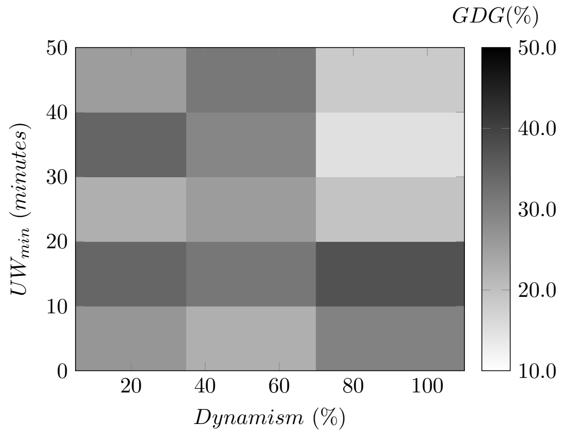
From Figure 10's results it can be concluded that tardiness is a dominant feature for instances with high urgency levels. It is unsurprising that a low ET typically results in re-optimization with a very limited view. The limited view implies that the algorithm does not recognize the cost of incoming urgent requests. This yields long travel times, but only a little tardiness. Another interesting result is that tardiness increases when buffer lengths increase. This is a direct consequence induced by buffering before conducting re-optimization and it results in a decrease in travel time.



(a)



(b)



(c)

Figure 11: GDGs obtained by the dynamic algorithm. Figures (a), (b) and (c) demonstrate that GDGs for the instances with different urgency levels and dynamism degrees are slightly affected by variations of these two request arrival characteristics.

In the context of dynamic problems, an algorithm performs well if request characteristics have a minimal impact upon its GDG. Figure 11 illustrates the primary achievement of the presented algorithm: that the GDG is only slightly affected by the dynamism and urgency of request arrivals. When dynamism increases for a low level of urgency, the GDG decreases. For a high level of urgency, decreases in the GDG continue until a certain degree of dynamism is reached (50%). For high urgency levels, the GDG increases for high degrees of dynamism while staying within the range of 15%-37%. The main reason why this gap increases slightly for very high degrees of dynamism is the more significant role data uncertainty plays. Another reason is increasing eligible request queue length. A longer queue may require more computation time than ET to achieve the same gap. Thus, eligible request queue length and ET function as two interrelated parameters which significantly affect the GDG. If ET is too small then a large gap is likely introduced. On the other hand, if ET is too large then the request queue's length may increase which will in turn also result in large gaps.

To conduct our experiments, we employed the instance generator introduced by van Lon et al. (2016), which provides dynamic PDPTW instances with a broad range of urgency levels and dynamism degrees. The main point of difference lies in that van Lon et al. (2016) assume that their algorithm “will never need to or can be interrupted during calculations and it will always give the best possible answer” (van Lon et al., 2016, p. 622). However, the validity of both these assumptions remains unproven. This paper proposes an interruptible algorithm. We also assessed its solution quality by calculating optimality gaps at each step (Figure 8) in addition to the gap at the end of the planning horizon (Figure 11). Given that van Lon et al. (2016) did not provide any evaluation information, we can not compare numerical results.

Impact of ET on the objective value. Another concern of this research is how ET impacts objective values. First, Figure 12 shows the dynamic algorithm’s objective values for instances with fixed urgency levels and multiple dynamism degrees. Second, Figure 13 details objective values for instances with fixed dynamism degrees and varying urgency levels.

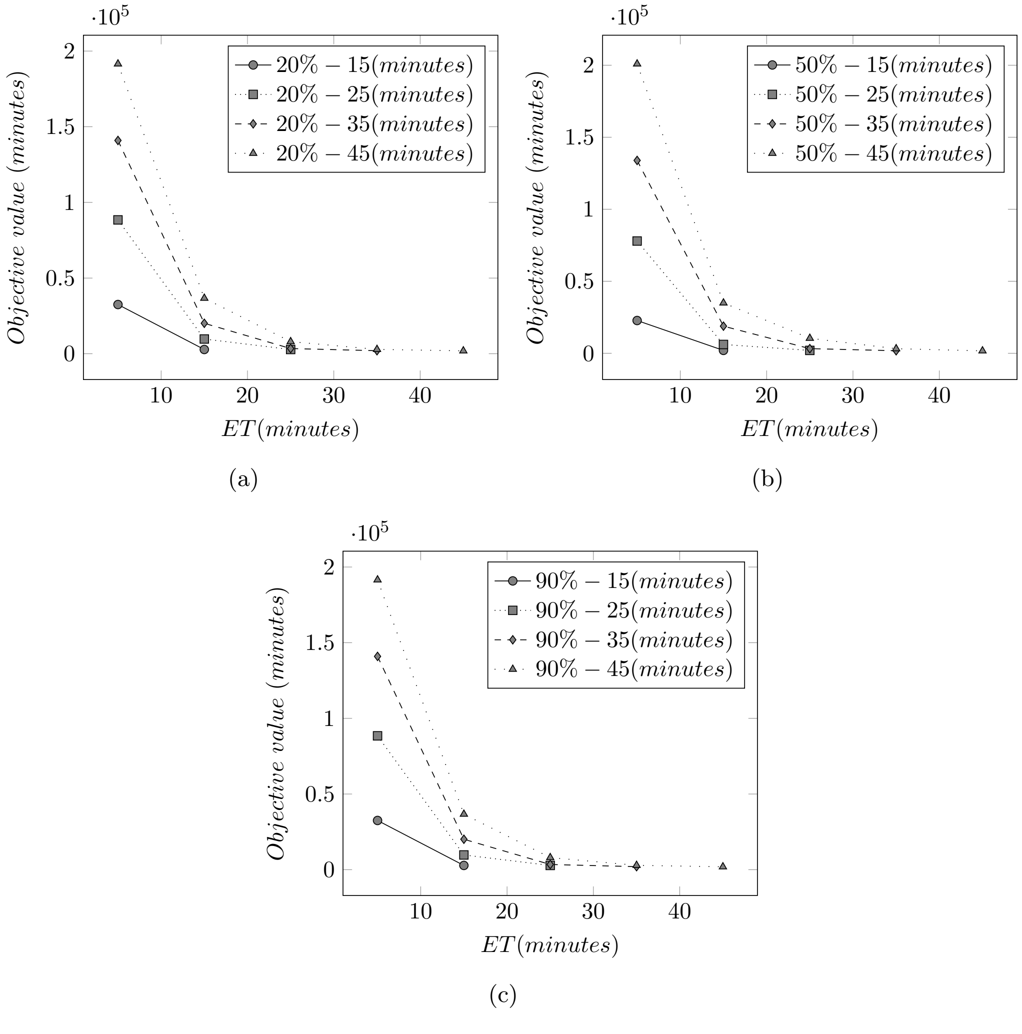


Figure 12: The impact of decreasing ET on the objective value for instances with fixed urgency levels and multiple degrees of dynamism.

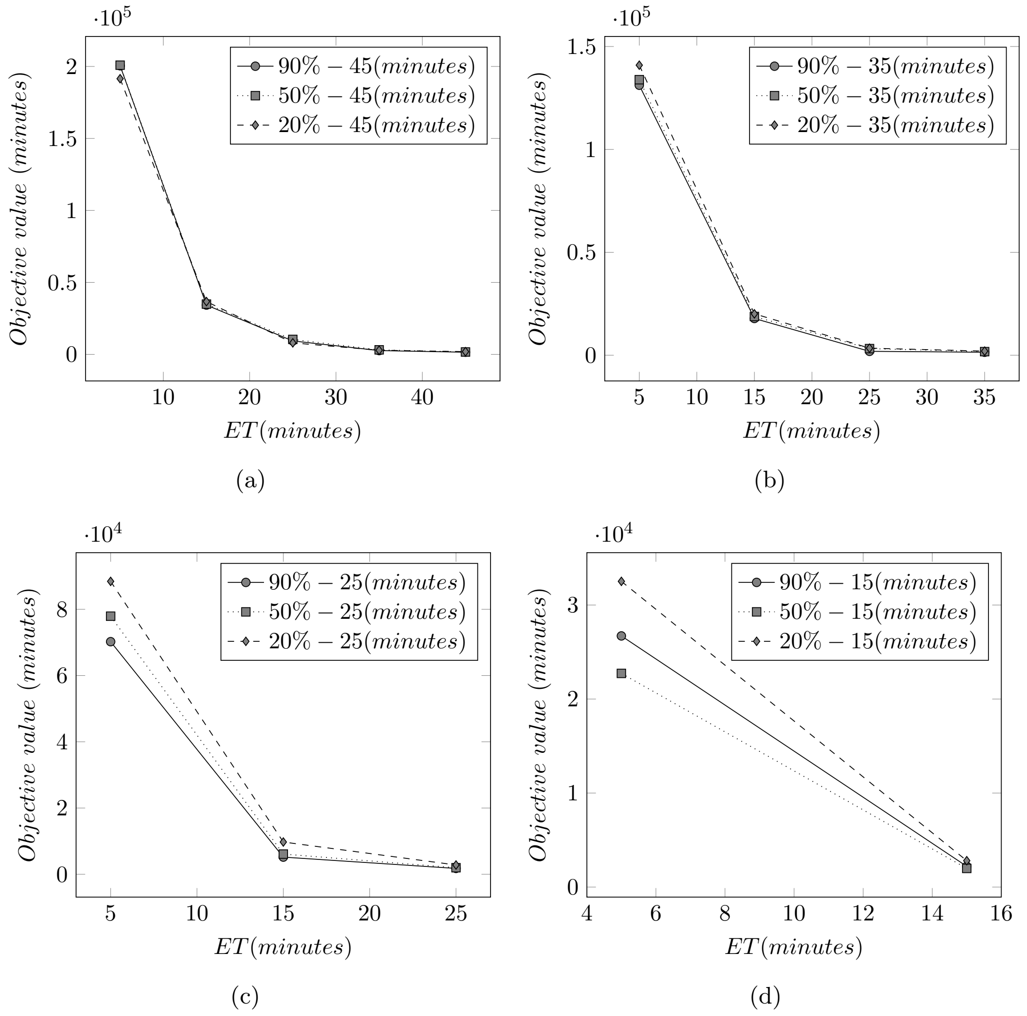


Figure 13: The impact of decreasing ET on the objective value for instances with fixed dynamism degrees and multiple urgency levels.

A high degree of dynamism results in greater input data uncertainty. Low urgency levels, meanwhile, result in longer request queues. Thus, the results illustrated throughout Figures 12 and 13 demonstrate how the highest quality (smallest GDG) solution is only achievable if the step length is at its maximum (UW_{min}). This is due to less data uncertainty and lengthier calculation times. Furthermore, for instances with low urgency, decreasing ET will increase both the objective value and GDG given that it introduces greater data uncertainty. Thus, Figure 13 implies that for low degrees of dynamism (20%) the objective value is more affected by decreasing the ET than by higher degrees of dynamism due to insufficient computation time.

6. Conclusions and future work

Almost all real-world logistics problems are characterised by the gradual release of information. Such problems consequently require efficient and repetitive decision making over time. Although over the last two decades considerable research effort has been allocated to dynamic problems, there is no agreed framework of methods and tools for comprehensive dynamic algorithm analysis. This paper is a response to these shortcomings and provides (i) an appropriate algorithm for the dynamic PDPTW (ii) a comprehensive study on the effect of frequent request arrivals on optimization efficiency and (iii) a performance assessment method.

This paper has introduced a periodic approach based on buffering for the dynamic PDPTW. Particular attention was given to the on-line construction of high-quality schedules, not only for each sub-problem but also at the end of the scheduling horizon. To this end, a MILP formulation for the static PDPTW was adapted to the dynamic PDPTW. It was shown that setting re-optimization step length equal to the maximum urgency of request arrivals outperforms all other step lengths. Next, the proposed periodic approach's performance was evaluated from both local and global perspectives. Additionally, the effect of frequent request arrivals on optimization efficiency has been studied and it was proven that the urgency level and dynamism degree of request arrivals as well as re-optimization frequency all impact the objective value. The results indicate that solution quality worsens when dynamism decreases and urgency increases. By contrast, solution quality improves when dynamism increases and urgency decreases. Nevertheless, it is noteworthy that the proposed algorithm's performance is only slightly affected by changes as regards dynamism and urgency. Moreover, solution quality worsens when step length decreases for instances with the same number of vehicles. This indicates that due to the inherent data uncertainty, short re-optimization step lengths mean that not all tasks can be efficiently handled by the available fleet.

Different aspects of the new dynamic algorithm have been studied with a view to building a comprehensive theory. Future research directions include the adaptation of existing approaches to large-scale real-world applications. This may be made possible by utilizing parallel computation for multiple buffering values. A possible direction would be to initiate the algorithm with a conservative buffer length and then adapt this length in accordance with the dynamism and urgency of request arrivals. Nevertheless, it may necessitate simultaneous real-time data processing to conclude on the periodic algorithm's best re-optimization frequency. Another interesting challenge for future research would be to consider vehicle capacity and fleet size to see how they impact operational decisions. Finally, we assumed request rejection was not allowed. However in real-world situations this constraint is more likely to be relaxed and a minimum value for the worst-case urgency level must therefore be determined to decide whether to accept or reject requests.

Another promising direction would concern further improving the scheduling algorithm which demands very fast and computationally efficient algorithms. The current state of the art with respect to the static version of the PDPTW is Slack Induction by String Removals (SISRs), a heuristic introduced in the forthcoming publication by Christiaens and Vanden Berghe (2020), which is currently available as a technical report. Given SISRs' good performance across a wide range of vehicle routing problems, it is likely that the heuristic would perform well once it has been adapted to accommodate the dynamic circumstances and objective we are interested in. However, proving so would require a thorough comparison against other methods such as the algorithms proposed by Bent and Van Hentenryck (2006); Pisinger and Røpke (2007); Lim and Zhang (2007); Nagata and Kobayashi (2010); Curtois et al. (2018). It would also be interesting to investigate the applicability of MILP to solve the problem at each re-optimization. There is a risk, however, of MILP solvers being unable to generate a feasible solution during each re-optimization step, in which case one would have to explore alternative strategies such as delaying or rejecting requests.

Acknowledgements

Wim Vancroonenburg is a postdoctoral research fellow at Research Foundation Flanders - FWO Vlaanderen. This research is funded by FWO as part of ORDinL (Operational Research and Data science in Logistics) project under the Strategic Basic Research (SBO) program. Editorial consultation provided by Luke Connolly (KU Leuven).

Appendix A. Terminology

θ_i	Input element i	S	Set of feasible solutions
ET	Execution time, re-optimization step length	Θ	Set of the PDPTW's instances
p_i	A task i's service time	θ	The PDPTW instance
$[st_i, dt_i)$	Time window of task i	R	Set of all requests
UW	Urgency window	T^p	Set of all pickup tasks
$[st_k, dt_k)$	Vehicle k 's availability time window	T^d	Set of all delivery tasks
τ	Scheduling horizon length	T	Set of all tasks ($T = T^p \cup T^d$)
V	Fleet of vehicles	V	Set of all vehicles
ε	All request arrival events	VK	Set of all undispatched vehicles
Ct_i	Task i's completion time	RL	Set of all eligible requests
ot_k	Vehicle k's overtime	UR_t	Set of unprocessed requests at time t
d_{ij}	Travel time from location i to j	CP_t,	Set of currently-processing requests at time t
l_i	Tardiness of task i	CE_t	Set of currently-executing requests at time t
ot_k	Overtime of vehicle k.	FR_t	Set of finished requests

Table A.3: List of notations

References

Angelelli, E., Bianchessi, N., Mansini, R., and Speranza, M. G. (2009). Short term strategies for a dynamic multi-period routing problem. *Transportation Research Part C: Emerging Technologies*, 17(2):106–119.

Attanasio, A., Bregman, J., Ghiani, G., and Manni, E. (2007). Real-time fleet management at ecourier ltd. In *Dynamic Fleet Management*, pages 219–238. Springer.

Attanasio, A., Cordeau, J.-F., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. (2012). *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Science & Business Media.

Barceló, J., Grzybowska, H., and Pardo, S. (2007). Vehicle routing and scheduling models, simulation and city logistics. In *Dynamic Fleet Management*, pages 163–195. Springer.

Beaudry, A., Laporte, G., Melo, T., and Nickel, S. (2010). Dynamic transportation of patients in hospitals. *OR Spectrum*, 32(1):77–107.

Bent, R. and Van Hentenryck, P. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893.

Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15.

Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2012). A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3):343–355.

Berbeglia, G., Pesant, G., and Rousseau, L.-M. (2011). Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 45(3):399–412.

Bock, S. (2010). Real-time control of freight forwarder transportation networks by integrating multimodal transport chains. *European Journal of Operational Research*, 200(3):733–746.

Borodin, A. and El-Yaniv, R. (2005). *Online computation and competitive analysis*. Cambridge university press.

Chang, M.-S., Chen, S.-R., and Hsueh, C.-F. (2003). Real-time vehicle routing problem with time windows and simultaneous delivery/pickup demands. *Journal of the Eastern Asia Society for Transportation Studies*, 5:2273–2286.

Cheung, B. K.-S., Choy, K., Li, C.-L., Shi, W., and Tang, J. (2008). Dynamic routing model and solution methods for fleet management with mobile technologies. *International Journal of Production Economics*, 113(2):694–705.

Christiaens, J. and Vanden Berghe, G. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science (to appear), available as TR, KU Leuven.*

Coslovich, L., Pesenti, R., and Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615.

Curtois, T., Landa-Silva, D., Qu, Y., and Laesanklang, W. (2018). Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, 7(2):151–192.

Dunke, F. and Nickel, S. (2016). A general modeling approach to online optimization with lookahead. *Omega*, 63:134–153.

Fabri, A. and Recht, P. (2006). On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological*, 40(4):335–350.

Ferrucci, F. and Bock, S. (2014). Real-time control of express pickup and delivery processes in a dynamic environment. *Transportation Research Part B: Methodological*, 63:1–14.

Fiegl, C. and Pontow, C. (2009). Online scheduling of pick-up and delivery tasks in hospitals. *Journal of Biomedical Informatics*, 42(4):624–32.

Fleischmann, B., Gnutzmann, S., and Sandvoß, E. (2004). Dynamic vehicle routing based on online traffic information. *Transportation Science*, 38(4):420–433.

Gan, Z., Tao, L., and Ying, Q. (2013). Automated guided vehicles dynamic scheduling based on annealing genetic algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11(5):2508–2515.

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

Ghiani, G., Manni, E., and Thomas, B. W. (2012). A comparison of anticipatory algorithms for the dynamic and stochastic traveling salesman problem. *Transportation Science*, 46(3):374–387.

Goel, A. and Gruhn, V. (2008). A general vehicle routing problem. *European Journal of Operational Research*, 191(3):650–660.

Gomes, R., de Sousa, J. P., and Dias, T. G. (2014). A grasp-based approach for demand responsive transportation. *International Journal of Transportation*, 2(1):21–32.

Grötschel, M., Krumke, S. O., Rambau, J., Winter, T., and Zimmermann, U. T. (2001). *Combinatorial online optimization in real time*. Springer.

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210.

Karami, F., Vancroonenburg, W., and Vanden Berghe, G. (2018). A dynamic scheduling approach to internal hospital logistics. In *The Second International Workshop on Dynamic Scheduling Problems, Poznań, Poland, June 26–28*, pages 57–62.

Kergosien, Y., Lente, C., Piton, D., and Billaut, J.-C. (2011). A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, 214(2):442–452.

Kilby, P., Prosser, P., and Shaw, P. (1998). Dynamic VRPs: A study of scenarios. *University of Strathclyde Technical Report*, pages 1–11.

Kouki, Z., Chaar, B. F., and Ksouri, M. (2009). Extended CNP framework for the dynamic pickup and delivery problem solving. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 61–71. Springer.

Larsen, A., Madsen, O. B., and Solomon, M. (2002). Partially dynamic vehicle routing-models and algorithms. *Journal of the Operational Research Society*, 53(6):637–646.

Li, J.-Q., Mirchandani, P. B., and Borenstein, D. (2009). Real-time vehicle rerouting problems with time windows. *European Journal of Operational Research*, 194(3):711–727.

Lim, A. and Zhang, X. (2007). A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 19(3):443–457.

Lund, K., Madsen, O. B., and Rygaard, J. M. (1996). Vehicle routing problems with varying degrees of dynamism (tech. rep.). *Lyngby, Denmark: IMM, The Department of Mathematical Modelling, Technical University of Denmark.*

Máhr, T., Srour, J., de Weerdt, M., and Zuidwijk, R. (2010). Can agents measure up? a comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research Part C: Emerging Technologies*, 18(1):99–119.

Mitrović-Minić, S., Krishnamurti, R., and Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685.

Mitrović-Minić, S. and Laporte, G. (2004). Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655.

Nagata, Y. and Kobayashi, S. (2010). A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. In *International Conference on Parallel Problem Solving from Nature*, pages 536–545. Springer.

Pisinger, D. and Røpke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Powell, W. B., Towns, M. T., and Marar, A. (2000). On the value of optimal myopic solutions for dynamic routing and scheduling problems in the presence of user noncompliance. *Transportation Science*, 34(1):67–85.

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154.

Psaraftis, H. N., Wen, M., and Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31.

Pureza, V. and Laporte, G. (2008). Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. *INFOR: Information Systems and Operational Research*, 46(3):165–175.

Renaud, J., Boctor, F. F., and Ouenniche, J. (2000). A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27(9):905–916.

Savelsbergh, M. W. and Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, 29(1):17–29.

Schyns, M. (2015). An ant colony system for responsive dynamic vehicle routing. *European Journal of Operational Research*, 245(3):704–718.

Secomandi, N. and Margot, F. (2009). Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations Research*, 57(1):214–230.

Speranza, M. G. (2018). Trends in transportation and logistics. *European Journal of Operational Research*, 264(3):830–836.

Swihart, M. R. and Papastavrou, J. D. (1999). A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3):447–464.

Ulmer, M. W., Mattfeld, D. C., and Köster, F. (2017). Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, 52(1):20–37.

van Lon, R. R., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., and Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3):614–624.

Vancroonenburg, W., Esprit, E., Smet, P., and Vanden Berghe, G. (2016). Optimizing internal logistic flows in hospitals by dynamic pick-up and delivery models. In *Proceedings of the 11th international conference on the practice and theory of automated timetabling, Udine, Italy, 23-26 August*, pages 371–383.

Yang, J., Jaillet, P., and Mahmassani, H. (2004). Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2):135–148.