Pickup and Delivery Problem with Time Windows: A Survey

Snezana Mitrović-Minić*, SFU

May, 1998

SFU CMPT TR 1998-12
ftp://fas.sfu.ca/pub/cs/techreports/1998

Abstract

This paper is a survey about the time constrained pickup and delivery problems and the methods used to solve them. The pickup and delivery problem is a problem of finding a set of optimal routes for a fleet of vehicles in order to serve transportation requests. Each transportation request is defined by a pickup location, a delivery location and a load. If the pickup and/or delivery location has a time interval within which it should be visited, the problem is known as the pickup and delivery problem with time windows. This time restricted version of the problem is the most researched because the majority of practical pickup and delivery problems deal with time restrictions.

Real life problems that can be modeled as pickup and delivery problems are: dial-a-ride problems, handicapped person transportation problems, courier company pickup and delivery problems, ...

This survey covers the most recent optimization methods and heuristics used for solving the pickup and delivery problem with time windows.

Acknowledgements

The author would like to thank supervisors Ramesh Krishnamurti (Simon Fraser University) and Gilbert Laporte (CRT, Université de Montréal) for substantial guidance and help. The supervisors have contirbuted directly to the production of this report.

*smitrovi@cs.sfu.ca

Contents

1 Introduction **3**

2 PDPTW **3**
2.1 Problem definition 3
2.2 Mathematical formulation of m-PDPTW 6
2.3 History of solution methods 9

3 Optimization methods for solving PDPTW **13**
3.1 Dynamic programming 13
3.2 Dantzig-Wolfe decomposition/Column generation 15

4 Heuristic methods for solving PDPTW **20**
4.1 Classic heuristic methods 20
4.2 Modern heuristic methods 31
4.3 Heuristics with finite bounds 36

5 Dynamic PDPTW **37**

6 Conclusions **38**

1 Introduction

This paper is a survey of approaches to the pickup and delivery problems with time windows. The author's intention was not to be exhaustive and describe all published techniques and methods, but to concentrate on several methods that, in the author's opinion, cover the main directions of achievements in the area.

The pickup and delivery problem is one of those combinatorial optimization problems that is attracting both researchers and practitioners. It has untied connections with practice. The problem statement and name comes from practice and the majority of the theoretical achievements originates in the development of dispatchers' decision support systems.

The pickup and delivery problem is a problem of finding a set of optimal routes for a fleet of vehicles in order to serve transportation requests. Each transportation request is defined by a pickup location, a delivery location and a load. If the pickup and/or delivery location has a time interval within which it should be visited, the problem is known as the pickup and delivery problem with time windows. This time restricted version of the problem is the most researched, which is not strange considering that almost all pickup and delivery problems in practice deal with some time restrictions. This is one of the reasons why this paper reports on the pickup and delivery problem with time windows.

The pickup and delivery problem is a generalization of the vehicle routing problem, which is a generalization of the traveling salesman problem, the well-known hard combinatorial optimization problem. Considering also that the problem in practice is, usually, of a large-scale, it is obvious why the problem is a challenge.

Many researchers in the last two decades worked on the problem and the significant achievements are reached. Still, there are areas and subproblems, yet, to be researched.

The purpose of this paper is to explain the problem and give an overview of the methods, developed for the problem.

Section 1 of the paper starts with the definition of the pickup and delivery problem with time windows and the related terms, and follows by presenting the mathematical programming formulation for the problem and a brief history of solution methods. Section 2, 3, and 4 are an overview of methods developed for solving the pickup and delivery problems with time windows.

Section 6 discusses the the dynamic pickup and delivery problems.

2 The pickup and delivery problem with time windows

2.1 Problem definition

The *pickup and delivery problem (PDP)* is a problem of finding a set of optimal routes, for a fleet of vehicles, in order to serve a set of transportation requests. Each vehicle from the fleet of vehicles, has a given capacity, a start location, and an end location. Each transportation request

is specified by a load to be transported, an origin, and a destination location. In other words, the pickup and delivery problem deals with the construction of optimal routes in order to visit all pickup and delivery locations and satisfy precedence and pairing constraints.

Precedence constraints deal with the restriction that each pickup location has to be visited prior to visiting the corresponding delivery location. *Pairing constraints* restrict the set of admissible routes such that one vehicle has to do both the pickup and the delivery of the load of one transportation request.

If the majority of pickup and delivery locations are distinct, the problem is said to be *many-to-many* PDP. In this paper we will address only those problems.

The *pickup and delivery problem with time windows (PDPTW)* deals with construction of optimal routes to satisfy transportation requests, precedence constraints, pairing constraints and time window constraints. Here, each transportation request has an additional characteristic - pickup and/or delivery location have to be visited in a desired time window. More precisely, the *time window constraint* forces a vehicle to be at a certain location at a time instant that is within the time interval (window) given by the customer.

If the pickup and delivery service will be done by one vehicle, the corresponding problem is called the single-vehicle pickup and delivery problem with time windows (1-PDPTW). If there is a fleet of vehicles available for the pickup and delivery, the problem is known as the multi-vehicle pickup and delivery problem (m-PDPTW).

A *customer* is a person that has a transportation request. A *trip* is a journey between one pickup location and its corresponding delivery location. A *stop location* is a common term for either pickup or delivery location.

Practical problems that can be modelled as pickup and delivery problems are:

- dial-a-ride problems (DARP)
- handicapped person transportation problems (HTP)
- courier company pickup and delivery problems (CCPDP)

The first two problems deal with transporting people, while the third problem deals with transporting messages and parcels. DARP is a multi-vehicle pickup and delivery problem with time windows (m-PDPTW). Most of the requests are known in advance. HTP is also m-PDPTW. In addition, while solving HTP one has to deal with different types of passengers where each type requires suitable space in a vehicle. The capacity constraints for both problems are tight. One more restriction in both problems is that the vehicle is not allowed to wait while carrying passengers. CCPDP problem is also m-PDPTW, but with loose, if any, capacity constraints. The other difference is that the time windows in CCPDP are much wider than in HTP and DARP. Transportation requests in DARP and HTP are characterized by desired pickup time or desired delivery time, and, eventually, maximal riding time. In CCPDP the customer is choosing time duration between pickup and delivery times. (On the basis of those times each algorithm is constructing time windows.) The fleet of vehicles is non-homogenous in practice,

for all problems. Multiple depots is a common characteristic in DARP and HTP, while CCPDP is usually characterized by no depot. In CCPDP each vehicle has starting and ending locations.

The term *service time* has two meanings in practice:

DARP, HTP: service time stands for desired pickup or desired delivery time, specified by a customer

CCPDP: service time is the time difference between pickup and delivery time, specified by a customer

The majority of real-life pickup and delivery problems are time restricted, consequently we have decided to research and report in this paper the main contributions and achievements in solving pickup and delivery problems with time windows.

In solving PDPTW the most studied *objective functions* are:

- Minimization of total time to service customers
- Minimization of total distance travelled
- Minimization of customers' inconvenience or 'dissatisfaction'
- Minimization of a weighted combination of:
  - time to service all customers and
  - total customers' inconvenience

Customer inconvenience is usually a linear function of a customer's waiting time and his/her excess riding time. Excess riding time is the difference between actual riding time and direct riding time.

The main *constraints* that a solution of the pickup and delivery problem have to satisfy are:

- time windows constraints
- capacity constraints where capacity can be multidimensional
- depot(s) locations constraints
- pairing (coupling) constraints
- precedence constraints
- resource constraints (drivers, vehicle types)

Additional constraints are possible: route duration restriction, drivers' lunch breaks, ...

The pickup and delivery problem *solution* consists of a set of routes and schedules for each vehicle. At any given time, a vehicle will be either in a *slack period* (i.e. idling) or in an *active period*. A *schedule block* is a continuous period of active vehicle time between any two successive slack periods. In recent literature a schedule block is referred by the term *mini-cluster*. A *route* is a sequence of vehicle stop locations. Route is sometimes called schedule sequence. A *time schedule*, or just *schedule*, is the list of times when the vehicle has to be at each stop location.

The process of *routing* is a process of finding routes i.e. a sequence of stops, for each vehicle. The process of *scheduling* is a process of finding time schedules for a given set of routes.

Finding an optimal schedule for a fixed route can be solved in $O(n)$ time. We will not discuss this problem in the paper.

The pickup and delivery problem with time windows (PDPTW) is a generalization of the vehicle routing problem with time windows (VRPTW). Furthermore, the vehicle routing problem is a generalization of the traveling salesman problem (TSP), the well-known problem in combinatorial optimization. Consequently PDPTW is an NP-hard problem.

It is worth mentioning that some pickup and delivery problems can be modelled as vehicle routing or even traveling salesman problems. Examples are:

- full-load pickup and delivery problem can be modelled as multiple traveling salesman problem with time windows (m-TSPTW),
- simple backhauling problem can be modelled by VRPTW with two resource constraints (capacity, time windows)
- single-vehicle pickup and delivery problem with time windows (1-PDPTW) is a constrained TSPTW where additional constraints are capacity and precedence constraints, etc.

2.2 Mathematical formulation of m-PDPTW

Suppose that we have n customers. Pickup location of customer i is node i and his/her delivery location is node $n+i$. Set V is a set of vehicles v. Starting and ending nodes of vehicle v are, respectively, $s(v)$ and $e(v)$. Set P^+ is a set of all pickup locations, P^- is a set of all delivery locations, and $P = P^+ \cup P^-$. Set N includes P and all starting and ending vehicle locations. Load at customer i is l_i units. Time window of the stop location i is $[a_i, b_i]$. For each two distinct stop locations, $t_{i,j}$ and $c_{i,j}$ represent direct travel time and travel cost from location i to location j.

Three types of variables are used in the mathematical formulation: binary flow variables $X^v_{i,j}$, time variables T_i, and load variables L_i. The binary flow variable $X^v_{i,j}$ has value 1 if vehicle v travels from node i to node j. Time variable T_i is the time when node i is serviced, and load variable L_i is equal to the load in the vehicle after servicing node i.

Mathematical programming formulation of a pickup and delivery problem with time windows is a non-linear integer program. If the objective is to minimize total cost, the program is given in Figure 1.

The description of the constraints are:

(2)-(5) Multi-commodity flow constraints

(2) One vehicle is leaving each pickup location and heading to unique stop location.

(3) Number of vehicles entering stop location i will leave location i.

(4) There is unique pickup location immediately after depot location for each vehicle.

(5) There is unique delivery location preceding depot location for each vehicle.

(6) Pairing constraints

(7) Precedence constraints

(8)-(10) Compatibility between routes and schedules

(11)-(13) Time window constraints

(14)-(16) Compatibility between routes and vehicles' capacity

(17)-(18) Capacity constraints

Mathematical programming problem formulation of a general pickup and delivery problem is given in [*Savelsbergh, Sol* (1995)].

If the problem is characterized by a single depot, the depot is labeled as node 0 when it is the starting location or as node $2n+1$ when it is the ending location. Other notation often used in the literature are: pickup node is denoted as $+i$, and delivery node is denoted as $-i$.

The references used in this subsection are [*Desrosiers, Dumas, Solomon, Soumis* (1995)], [*Dumas, Desrosiers, Soumis* (1991)], [*Savelsbergh, Sol* (1995)].

$$\text{min} \quad \sum_{v \in V} \sum_{i,j \in N} c_{i,j} X^v_{i,j} \tag{1}$$

$$\text{subject to} \quad \sum_{v \in V} \sum_{j \in N} X^v_{i,j} = 1, \qquad i \in P^+ \tag{2}$$

$$\sum_{j \in N} X^v_{i,j} - \sum_{j \in N} X^v_{j,i} = 0, \qquad i \in P,\ v \in V \tag{3}$$

$$\sum_{j \in P^+} X^v_{s(v),j} = 1, \qquad v \in V \tag{4}$$

$$\sum_{i \in P^-} X^v_{i,e(v)} = 1, \qquad v \in V \tag{5}$$

$$\sum_{j \in N} X^v_{i,j} - \sum_{j \in N} X^v_{j,n+i} = 0, \qquad i \in P^+,\ v \in V \tag{6}$$

$$T^v_i + t^v_{i,n+i} \leq T^v_{n+i}, \qquad i \in P^+,\ v \in V \tag{7}$$

$$X^v_{i,j} = 1 \ \Rightarrow\ T^v_i + t^v_{i,j} \leq T^v_j, \qquad i,j \in P,\ v \in V \tag{8}$$

$$X^v_{s(v),j} = 1 \ \Rightarrow\ T^v_{s(v)} + t^v_{s(v),j} \leq T^v_j, \qquad j \in P^+,\ v \in V \tag{9}$$

$$X^v_{i,e(v)} = 1 \ \Rightarrow\ T^v_i + t^v_{i,e(v)} \leq T^v_{e(v)}, \qquad i \in P^-,\ v \in V \tag{10}$$

$$a_i \leq T^v_i \leq b_i, \qquad i \in P,\ v \in V \tag{11}$$

$$a_{s(v)} \leq T^v_{s(v)} \leq b_{s(v)}, \qquad v \in V \tag{12}$$

$$a_{e(v)} \leq T^v_{e(v)} \leq b_{e(v)}, \qquad v \in V \tag{13}$$

$$X^v_{i,j} = 1 \ \Rightarrow\ L^v_i + l_j = L^v_j, \qquad i \in P,\ j \in P^+,\ v \in V \tag{14}$$

$$X^v_{i,j} = 1 \ \Rightarrow\ L^v_i - l_{j-n} = L^v_j, \qquad i \in P,\ j \in P^-,\ v \in V \tag{15}$$

$$X^v_{s(v),j} = 1 \ \Rightarrow\ L^v_{s(v)} + l_j = L^v_j, \qquad j \in P^+,\ v \in V \tag{16}$$

$$L^v_{s(v)} = 0, \qquad v \in V \tag{17}$$

$$l_i \leq L^v_i \leq Q^v, \qquad i \in P^+,\ v \in V \tag{18}$$

$$X^v_{i,j} \in \{0,1\}, \qquad i,j \in N,\ v \in V \tag{19}$$

Figure 1: Mathematical formulation of m-PDPTW

2.3 History of solution methods

Surveys on the pickup and delivery problems can be found in [*Bodin, Golden, Assad, Ball* (1983)], [*Savelsbergh, Sol* (1995)], [*Fisher* (1995)], [*Desrosiers, Dumas, Solomon, Soumis* (1995)], [*Powell, Jaillet, Odoni* (1995)], [*Assad* (1988)].

The research, primarily on practical problems, started in the late 1970s. The problem has been first examined by Wilson et al. The authors have developed real-time algorithms for dial-a-ride systems operating in Haddonfield, NJ and Rochester, NY. This research set up some fundamental concepts for solving pickup and delivery problems: building tours through (sequential) insertion of customers, and the general form of the objective function.

The early work on ***mathematical formulation*** of the problem was done in the late 1970s. The formulation has since changed and improved. See [*Savelsbergh, Sol* (1995)] for the latest work on the formulation of the general pickup and delivery problems.

Exact methods for solving pickup and delivery problems were first published in [*Psaraftis* (1983a)]. The author has worked on two versions of the single-vehicle dial-a-ride-problem: the immediate-request problem [*Psaraftis* (1980)] and the problem with time windows [*Psaraftis* (1983a)]. The dynamic programming algorithms were developed.

Almost two decades later a new exact method was developed by [*Dumas, Desrosiers, Soumis* (1991)]. The new technique is based on the Dantzig-Wolfe decomposition/column generation approach. Aimed to solve a specific class of multi-vehicles pickup and delivery problems, the method is much more robust than its counterpart. The algorithm is easy adaptable to handle different objective functions, multiple depots, and non-homogenous fleet of vehicles.

Other papers in the area of the time constrained pickup and delivery problems deal with ***heuristic algorithms***. After the early 1970s, for more than two decades, researchers focused on three main ideas: decomposition methods, insertion heuristics and local search methods.

Decomposition methods range from relatively simple 'cluster-first, route-second' idea known as the two phase method, to very complex method of mini-clustering. Examples include: Jaw et al. (1982) who develop a three phase method by introducing decomposition in time, and [*Sexton, Bodin* (1985a), *Sexton, Bodin* (1985b)] who develop algorithms based on Benders decomposition. [*Dumas, Desrosiers, Soumis* (1989b)] improved the decomposition approach by introducing and developing the method based on mini-clustering.

Insertion heuristics for the pickup and delivery problem started with the development of sequential insertion heuristics. A few years later, the parallel insertion algorithm was developed. The idea and the algorithm is improved by [*Jaw, Odoni, Psaraftis, Wilson* (1986)] while developing ADARPTW package for dial-a-ride problem. Further algorithm advances were done by [*Madsen, Ravn, Rygaard* (1995)] in package named REBUS.

Local search algorithms have been studied by [*Savelsbergh* (1990)], [*Van Der Bruggen, Lenstra, Schuur* (1993)], [*Thompson, Psaraftis* (1993)].

New approaches based on modern heuristics and metaheuristics were applied to pickup and delivery problems: [*Toth, Vigo* (1995), *Toth, Vigo* (1997)] have worked on tabu threshold procedure, [*Shen, Potvin, Rousseau, Roy* (1995)] have developed a system based on automated learning and neural networks, [*Van Der Bruggen, Lenstra, Schuur* (1993)] have used simulated annealing technique, [*Potvin, Rousseau* (1992)] have applied yet another artificial intelligence method: constraint-directed search.

A summary of history of solution methods is given in the following four tables. The first two tables show methods used for solving single-vehicle pickup and delivery problem with time windows (1-PDPTW), and the last two tables illustrate research in solving multi-vehicle pickup and delivery problem with time windows (m-PDPTW).

Optimization algorithms			
Authors	*Problem*	*Solution method*	*Results*
Psaraftis (1983a)	DARP	Forward **dynamic programming**, $O(n^2 3^n)$	10 requests
Desrosiers, Dumas, Soumis (1986)	DARP	Forward **dynamic programming** approach. State elimination criteria very effective when TWs are tight and vehicle capacities are small.	40 requests, 6 seconds

Table 1: Optimization methods for solving 1-PDPTW

Heuristic algorithms			
Authors	*Problem*	*Solution method*	*Results*
Wilson et al. (1971, 1976, 1977)	DARP	Early work on real-time DARP. **Insertion procedure**.	
Sexton, Bodin (1985)	DARP	Heuristic version of **Benders decomposition** applied to a mixed 0-1 non-linear programming formulation. Decomposition separates the routing and scheduling component.	20 requests, 20 seconds
Van Der Bruggen, Lenstra, Schuur (1993)		**Local search procedure**: based on Lin-Kernighan variable-depth arc-exchange method for TSP. Solutions either optimal or within 1% of optimal, in reasonable time.	50 requests
Van der Bruggen, Lenstra, Schuur (1993)		**Simulated annealing procedure.** High quality solutions. Larger CPU time.	

Table 2: Heuristics for solving 1-PDPTW

Optimization algorithms			
Authors	*Problem*	*Solution method*	*Results*
Desrosiers, Dumas, Soumis (1991)		**Dantzig-Wolfe decomposition** approach embedded in branch and bound tree. Master problem is set partitioning, subproblem is constrained shortest path problem.	55 requests, 22 vehicles

Table 3: Optimization methods for solving m-PDPTW

Heuristic algorithms			
Authors	*Problem*	*Solution method*	*Comp.results*
Hung, Chapman, Hall, Neigut (1982)	DARP	Sequential **insertion** procedure	
Jaw, Odoni, Psaraftis, Wilson (1982)	DARP	**Decomposition** - three phase approach: 1. Grouping phase (time horizon division) 2. Clustering phase (space) 3. Routing phase	
Roy, Chapleau, Ferland, Lapalme, Rousseau (1983)	DARP	Parallel **insertion** procedure	
Bodin, Sexton (1980)	DARP	**Decomposition**: cluster requests and than find routes and schedules by authors' single-vehicle algorithm. **Improvement**: by moving requests out from bad routes.	85 requests, 7 vehicles
Jaw, Odoni, Psaraftis, Wilson (1986)	DARP	**Insertion** to be done in cheapest feasible way. If there is no feasible insertion initiate new route. Terms: active period, slack time. Have stored: possible backward and forward time shifts for each stop location.	250 requests, 14 vehicles; 20 seconds. 2600 requests, 20 vehicles, 12 minutes
Psaraftis (1986)	DARP	**Tests** on Jaw et al. (1982) three phase approach and Jaw et al. (1986) insertion heuristic. Jaw et al. (1982) method good for finding initial solution.	2600 cust., 20 vehicles
Dumas, Desrosiers, Soumis (1988, 1989)	HTP	**Decomposition**: Improved cluster-routing algorithm by moving a part of clustering into routing: **Mini-clustering** followed by optimal routing. Mini-clustering is m-PDPTW problem solved by: routes, constructed by some heuristic, are divided into mini-clusters. Routing problem is m-TSPTW problem. It is solved optimally by delayed column generation.	190 requests, 31 vehicle, 85 mini-clus. 880 requests, 53 vehicles, 282 mini-clus. in several time-slices
Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991)	HTP	**Mini-clustering** followed by optimal routing. Parallel insertion heuristic directs mini-clustering.	
Ioachim, Desrosiers, Dumas, Solomon (1991)	HTP	**Mini-clustering** followed by optimal routing. Column generation is used for solving both m-PDPTW and m-TSPTW.	3585 requests, 1 hour
Potvin, Rousseau (1992)	DARP	**Constraint-directed search**. 5-15% better solutions than Jaw et al. (1986) but 2-5 time slower.	
Madsen, Ravn, Rydgaard (1993)	DARP	Improved **insertion** heuristic of Jaw et al. (1986)	300 requests, 24 vehicles, 10 seconds
Thompson, Psarftis (1993)		**Cyclic transfers** improvement procedure.	
Shen, Roy, Rousseau, Potvin (1995)	CCPDP	Automated learning: **Neural networks**	
Toth, Vigo (1996,1997)	HTP	Parallel **insertion** algorithm + Tour improvement (refining) procedure: **Tabu Treshold algorithm**	

Table 4: Heuristics for solving m-PDPTW

3 Optimization methods for solving PDPTW

Optimization method is aimed to find an optimal solution to the problem. In the context of pickup and delivery problems, an optimization method has to find a set of routes such that the value of the objective function is the best among all possible sets of routes. An optimization method can also be called an optimal method or an exact method.

Optimization methods developed for solving PDPTW use the dynamic programming or the Dantzig-Wolfe decomposition approach.

Optimization algorithms				
Method	*Problem*	*Years*	*Authors*	*Results*
Dynamic programming	1-PDPTW (DARP)	1983-	Psaraftis (1983)	10 requests
	1-PDPTW (DARP)	1986	Desrosiers, Dumas, Soumis (1986)	40 requests
Dantzig-Wolfe decomposition/Column generation	m-PDPTW (tight capacity constraints)	1991	Desrosiers, Dumas, Soumis (1991)	55 requests, 22 vehicles

Table 5: Optimization methods for PDPTW

3.1 Dynamic programming

Dynamic programming is an intelligent enumeration of all feasible solutions of an optimization problem [*Papadimitrou, Steiglitz* (1982)]. The idea is to work backwards from the last decision to the earlier ones. Suppose that, in order to solve an optimization problem, one has to make a sequence of decisions, say $D_1, D_2, ..., D_n$. Then if the sequence leads to an optimal solution, the last k decisions, $D_{n-k+1}, D_{n-k+2}, ..., D_n$ must be optimal. This fact is known as *principle of optimality*. Dynamic programming is based on this principle. A construction of a dynamic programming algorithm consists of: (1) decomposition of the problem into stages (states) at which decisions take place and (2) definition of recurrence formula that leads from one state to the other. The effectiveness of dynamic programming is based on the reduction of the state space and on the reduction in the number of state transitions.

Dynamic programming was the first optimization method applied on the pickup and delivery problems. The first published dynamic programming method for solving 1-DARP was developed by [*Psaraftis* (1983a)]. Both pickup *and* delivery times are considered. No capacity or priority constraints were taken into account. The travel times matrix is not required to be symmetric. The *objective* is to minimize the time to service all customers. The author states that in 1-DARPTW, the number of feasible solutions is so limited that the variation in the optimal schedule under alternative objective functions may not be significant. This is true in the case of the single-vehicle problem in the presence of narrow time windows.

The dynamic programming algorithm has been developed on the basis of [*Psaraftis* (1980)] an algorithm which was aimed in solving the single-vehicle immediate-request dial-a-ride problem

i.e. 1-PDP.

Backward dynamic programming for 1-PDP. In [*Psaraftis* (1980)] the author solves 1-PDP by dynamic programming, using the backward recursion scheme. In this problem, each customer requests a service by phone, and the call is recorded in a list, by chronological order. After a certain period of time the list of customers is closed and a vehicle tour is designed in order to serve all known customers i.e. all customers from the chronological list. No new customers are eligible for insertion.

An interesting innovation of the method is the use of a *maximum position shift*. A *pickup position shift* is the difference between its position in the chronological list and its actual position in the sequence of customer pickups. The *delivery position shift* is defined similarly. In order to guarantee 'good' service to all customers the author has introduced a bound for both the pickup position shift and the delivery position shift. The bound is the maximum position shift constant (MPS). Thus MPS suppresses indefinite deferment of serving 'hard' requests. Small MPS values impose that each customer will be served 'better' i.e. 'more fairly'. It is to be noted that the low values of MPS significantly reduces the set of feasible solutions. The *objective function* for this immediate-request problem was to minimize a weighted combination of total time to service all customers and total customers' 'dissatisfaction'. 'Dissatisfaction' of each customer was a linear function of the customer's waiting time and his/her riding time.

The *state vector* used in the algorithm is $(L, k_1, k_2 ... k_n)$ where L is the current vehicle location and k_i is the status of customer i. The status of a customer can have a value of 3 if the customer has not been picked up, a value of 2 if the customer has been picked-up, and a value of 1 if the customer has been delivered. The feasibility of each state is checked by examining the violation of precedence constraints, pairing constraints, capacity constraints, and MPS constraints.

The given dynamic programming algorithm is based on the dynamic programming method for solving TSP.

The algorithm complexity is $O(n^2 3^n)$ for n customers.

Experiments have shown that a problem with nine customers can be solved in less than 10 minutes in the worst case i.e. when MPS is large and capacity constraints are not considered. The running time for seven customers was 47 seconds. When the capacity constraints was 4, the MPS is 3, instances with seven customers was solved in 25 seconds. The experiments have shown that the computational effort for solving PDP is asymptotically lower than the effort needed for solving TSP.

Forward dynamic programming for 1-PDPTW. The method for solving 1-PDPTW is based on the given method for solving 1-PDP. In the method for solving 1-PDPTW [*Psaraftis* (1983a)] does not use Maximum Position Shift constant. The other main difference was that [*Psaraftis* (1983a)] algorithm is a *forward recursion* scheme. This was due the fact that the backward approach cannot keep track of time from a specified time instant onward.

Algorithm complexity is the same i.e. $O(n^2 3^n)$.

A few years later, yet another dynamic programming algorithm for solving the single-vehicle pickup and delivery problem was reported [*Desrosiers, Dumas, Soumis* (1986)]. The scheme also used a forward dynamic programming. The *objective* was to minimize the total distance travelled. The algorithm solves problems of up to 40 customers in just a few seconds. Its effectiveness is largely due to the use of nine efficient state elimination criterias. These criterias have reduced the average number of examined states, for problems with 31-40 customers, to 2% of the total number of states.

The *state* used in the algorithm is (S, i) where S is a subset of all stop locations. The state is feasible if there is a feasible path through all nodes in S ending in i. The state (S, i) is post-feasible if there is a feasible path starting at i and visiting all nodes not in S. Post-feasibility is checked by examining if each unvisited location can be visited immediately after i by not violating time windows constraints. Two labels were recorded for each path of each state (arrival time, total distance).

The algorithm has shown very good results when TWs were tight and vehicle capacities were small. The memory requirements were modest, due to effective state elimination.

3.2 Dantzig-Wolfe decomposition/Column generation

Dantzig-Wolfe (DW) decomposition is a technique for solving large-scale linear programming problems. It was developed by Dantzig and Wolfe in mid-1950s. A few decades later, researchers found that the technique could be very effective for solving large-scale combinatorial optimization problems. The idea was to embed the Dantzig-Wolfe decomposition method in a branch-and-bound tree. The advantage of the Dantzig-Wolfe decomposition, in the context of this approach, is that it can give very tight lower bounds for integer programming problems. The result is a small branch-and-bound tree. Hence, despite the fact that DW decomposition is time consuming, the branch and bound method will need reasonable computing time.

The main idea behind the Dantzig-Wolfe decomposition method lies in the fact that any linear programming problem

$$\begin{array}{lll} (P) & \max & cx \\ & \text{subject to} & Ax = b \\ & & l \leq x \leq u \end{array}$$

could be written in the form of so-called *master problem*

$$\begin{array}{lll} (\tilde{P}) & \max & \tilde{c}\tilde{x} \\ & \text{subject to} & \tilde{A}\tilde{x} = \tilde{b} \\ & & \tilde{x} \geq 0 \end{array}$$

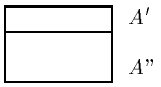
where matrix $\tilde{A}$ has fewer rows but usually many more columns than A, and where general bounds on variables are changed into nonnegativity constraint. The master problem is in a

form that can be solved effectively by the simplex method. The revised version of the simplex method will be used which does not require matrix $\tilde{A}$ and $\tilde{c}$ to be given explicitly. Each step of the revised simplex method requires one column of matrix $\tilde{A}$ to be generated. The combination of the revised simplex method and the column generation on request is known as the *delayed column generation* technique.

In the following paragraphs we will give concise explanations of:

- Master problem construction,
- Delayed column generation technique,
- Main steps in the Dantzig-Wolfe decomposition/Column generation method, and
- Application of these techniques to pickup and delivery problems.

Generation of master problem $\tilde{P}$. Let partition the $(m \times n)$ matrix A of problem P into an $(m' \times n)$ matrix A' and an $(m" \times n)$ matrix $A"$ in the following way:



Then problem P can be written as a problem Q:

$$
\begin{array}{lll}
(Q) & \max & cx \\
 & \text{subject to} & A'x = b' \\
 & & A"x = b" \\
 & & l \leq x \leq u
\end{array}
$$

Perold [*Chvatal* (1983)] has suggested that it is desirable to partition A such that the set of constraints is partitioned into:

- a small set of (possibly hard) constraints $A'x = b'$ and
- a (possibly large) set of easy constraints $A"x = b"$

Let problem Q decompose into two problems: problem Q_1 containing the first two lines of problem Q

$$
\begin{array}{lll}
(Q_1) & \max & cx \\
 & \text{subject to} & A'x = b'
\end{array}
$$

and problem Q_2 containing the last two lines of problem Q

$$
\begin{array}{ll}
(Q_2) & A"x = b" \\
 & l \leq x \leq u
\end{array}
$$

A theorem from linear programming theory states that each vector x which satisfies Q_2 can be represented as a linear combination of vectors v_k and w_k:

$$(LC) \qquad x = \sum_{k=1}^{M} r_k v^k + \sum_{k=1}^{N} s_k w^k$$

where r_k and s_k are nonnegative and will be new problem variables. Vectors v_k and w_k represent the set of all vertices of the polyhedron defined by Q_2, and there is a technique for their calculation.

If x in Q_1 is substituted by LC, the problem Q_1 will, after several transformations, become the master problem $\tilde{P}$.

$$\begin{array}{lll} (\tilde{P}) & \max & \tilde{c}\tilde{x} \\ & \text{subject to} & \tilde{A}\tilde{x} = \tilde{b} \\ & & \tilde{x} \geq 0 \end{array}$$

This master problem $\tilde{P}$ is a linear programming problem that can be solved by the simplex method.

Problem Q_2 will be called *subproblem.*

Unfortunately, not all difficulties are solved by the given decomposition. A problem that arises is that the number of vectors v_k and w_k can be huge and, as such, it is often impossible to generate them all. Thus, matrix $\tilde{A}$ cannot be given explicitly. The resolution lies in the use of the revised simplex method and the generation of one new column in each iteration of the revised simplex method.

Delayed column generation or, just, column generation. The main purpose of the delayed column generation scheme is solving a special class of linear programming problems by the simplex method. The class of problems includes the linear programming problems characterized by an immense number of variables and therefore an immense number of columns. Very often in practice, these columns cannot be given explicitly. The revised version of the simplex method, fortunately, does not require all columns in each of its iterations. If this advantage is to be used, an obstacle will be the generation of a new column. This new column will be the entering column in the current iteration of the revised simplex method. The next iteration of the simplex method will be preceded by the generation of another column. This is continued until optimal basis is obtained. The name of this scheme is *delayed column generation.*

The generation of one column is the generation of a basic feasible solution of subproblem. Depending on the subproblem structure, algorithms for generating new columns can vary from algorithms for solving a simple system of two inequalities (in solving the cutting-stock problem) to algorithms for solving complex optimization problems (like constrained shortest paths problem when solving pickup and delivery problems).

(For further details refer to [*Chvatal* (1983)].)

Main steps of the Dantzig-Wolfe decomposition/Column generation technique in solving a problem are shown in Figure 2:

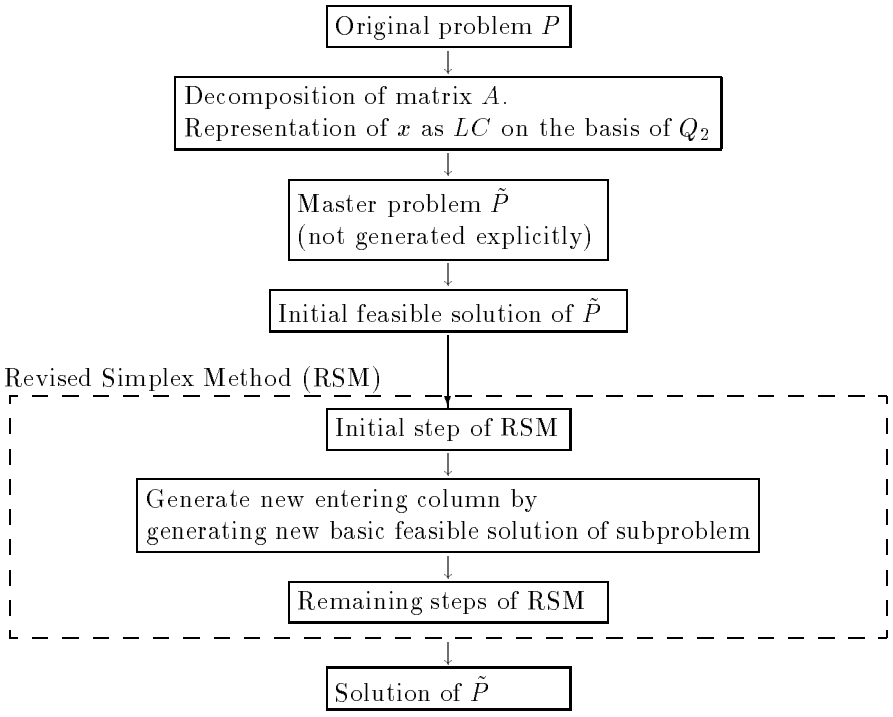


Figure 2: Dantzig-Wolfe decomposition/Column generation method

Pickup and delivery problems. DW decomposition is applied by Desrosiers et al. on a variety of time constrained vehicle routing problems, including TSPTW and PDPTW.

The main idea is in the following steps:

- Make a mathematical formulation of PDPTW, which is a non-linear integer program.
- Do a linearization of a non-linear program by the introduction of large parameters. The result is a linear integer program IP.
- Make a linear relaxation of IP. The result is a linear program LP.
- Apply the Dantzig-Wolfe decomposition to LP. The result will be a Master problem (MP) and a Subproblem (SP).
- Solve the master problem by the delayed column generation method. Column generation will be done by solving the subproblem.

Now, if we embed this whole procedure in a branch and bound tree in order to get integer solutions, the given method is an exact method for solving PDPTW.

Refer again to the mathematical programming formulation on Page 8. Application of the Dantzig-Wolfe decomposition will form the master problem from the objective function (1) and constraint (2), while the subproblem consists of a modified objective function and constraints (3)-(19). Note that the subproblem can be partitioned by vehicle. The master problem after several transformations will become the linear relaxation of a set partitioning type problem. The set to be partitioned is the set of all possible routes.

$$\begin{aligned} \min \quad & \sum_r c_r x_r \\ & \sum_r a_{ir} x_r = 1 \\ & \sum_r x_r = |M| \end{aligned}$$

And now the situation is the same as already explained in the general case - the number of all possible routes is generally too large to allow exhaustive enumeration. The delayed column generation method is used to solve the master problem. The admissible routes (columns) are generated by solving constrained shortest path problems.

The solution of the master problem is an excellent lower bound for the initial PDPTW. In order to get integer solutions to the initial PDPTW, the whole procedure is embedded into a branch and bound tree in order to get integer solution of PDPTW.

[*Dumas, Desrosiers, Soumis* (1991)] presented this kind of exact method for solving some specific instances of m-PDPTW when transporting goods. The given algorithm uses the column generation scheme with a constrained shortest path problem as a subproblem. The algorithm works well when the demand at each customer is large i.e. when capacity constraints are tight (restrictive). Problem instances for algorithm testing have demands similar to this example: 40% of requests has demand for full load, 40% half load, 20% one third load. On the other side, the approach is very robust in the sense that the model can be adapted easily to handle different objective functions, multiple depots, non-homogenous fleet of vehicles. Other problem characteristics were: existence of pickup *and* delivery time windows, and the objective to minimize total travel costs. The underlying *network* of the problem is defined by:

- Set of nodes includes all pickup and delivery locations and all depot locations.
- Set of network arcs is equal to the set of all admissible arcs. The set of admissible arcs is determined in the preprocessing step called *network construction.*

A preliminary step to network construction is the *reduction of time windows.* Time windows reduction is based on the fact that the partial paths

$(n+i \rightarrow 2n+1)$ i.e. (delivery location $\rightarrow$ depot) and
$(i \rightarrow n+i \rightarrow 2n+1)$ i.e. (pickup location $\rightarrow$ matching delivery location $\rightarrow$ depot)

have to be admissible for any schedule. In the case of single-depot the time reduction formulas are alike:

$$\begin{aligned} b_{n+i} &= min\{b_{n+i}, b_{2n+1} - t_{n+i,2n+1}\} \\ b_i &= min\{b_i, b_{n+i} - t_{i,n+i}\} \\ a_i &= max\{a_i, a_0 + t_{0,i}\} \\ a_{n+i} &= max\{a_{n+i}, a_i + t_{i,n+i}\} \end{aligned}$$

Network construction of admissible arcs: The determination of admissible arcs is based on the following constraints: time windows constraints, priority constraints, vehicle capacity constraints, and pairing constraints.

The described DW decomposition method, embedded in branch and bound tree, was tested on problems involving 55 requests and 22 vehicles. The results were very good.

4 Heuristic methods for solving PDPTW

Due to the fact that the PDP is NP-hard problem, combined with the reality that practical PDPs are very large, having hundreds and even thousand of requests to serve, there is no much hope for finding an optimal algorithm that will work acceptably fast in practice. This is the main reason why the majority of research in the area is focused on finding an algorithm that will find 'good' solution in reasonably short time. These kind of algorithms are called *heuristics*.

Heuristic algorithms have been applied in solving pickup and delivery problems since the beginning of the 1970s.

The main classification of heuristics for this type of routing and scheduling problems are:

- classic heuristic methods
  - tour construction heuristics
  - tour improvement heuristics
- modern heuristic methods
  - metaheuristics

4.1 Classic heuristic methods

4.1.1 Tour construction heuristics

Tour construction algorithms build (construct) a set of feasible routes, starting from information that define pickup and delivery problem.

Tour construction algorithms can be divided into two groups:

Heuristic algorithms				
Method	*Problem*	*Years*	*Authors*	*Results*
CLASSIC HEURISTIC METHODS				
Construction heuristics				
Decomposition methods				
Three phase method: grouping in time, clustering, routing.	m-PDPTW (DARP)	1982	Jaw, Odoni, Psaraftis, Wilson (1982)	
Clustering followed by Benders decomposition: routing and scheduling.	m-PDPTW (DARP)	1980-1985	Sexton, Bodin (1980,1985)	85 requests, 7 vehicles
Mini-clustering followed by routing solved by column generation. Time-slicing.	m-PDPTW (HTP)	1989-1991	Desrosiers, Dumas, Soumis (1989) Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991) Ioachim, Desrosiers, Dumas, Solomon (1991)	190 requests, 22 vehicles 880 requests, 53 vehicles
Insertion heuristics				
Insertion procedure	m-PDPTW	1986 1997	Jaw, Odoni, Psaraftis, Wilson (1986) Toth, Vigo (1997)	
Improvement heuristics				
Local search improvement procedure	m-PDPTW	1983	Bodin, Sexton (1983)	
Variable-depth arc exchange procedure	1-PDPTW	1993	Van der Bruggen, Lenstra, Schuur (1993)	50 requests
Cyclic transfers	m-PDPTW	1993	Thomposon, Psaraftis (1993)	
MODERN HEURISTIC METHODS				
Metaheuristics				
Constraint-directed search	m-PDPTW (DARP)	1992	Potvin, Rousseau (1992)	
Simulated annealing	1-PDPTW	1993	Van der Bruggen, Lenstra, Schuur (1993)	
Neural networks	m-PDPTW	1995	Chen, Roy, Rousseau, Potvin (1995)	
Tabu threshold improvement procedure	m-PDPTW	1996	Toth, Vigo (1996)	

Table 6: Heuristics for PDPTW

- decomposition methods
- insertion procedures

Decomposition methods

Decomposition methods for 1-PDPTW are based on the idea of dividing a problem into two phases: routing and scheduling. Decomposition methods for m-PDPTW has one preliminary phase - clustering. *Clustering* is a partitioning of requests into subsets each of which will be served by one vehicle, and is the difficult part. After clustering is done, the routing can be solved optimally or by a heuristic. But if clustering is not good enough, routing by itself cannot correct it.

Some decomposition methods are simple 'cluster-first, route-second' procedures, others are more sophisticated mathematical programming approaches that try to move some of the clustering part of the problem into the routing part.

Some of the decomposition algorithms includes the initial phase of dividing a problem horizon into time slices, and partitioning the set of requests corresponding to them. This phase is usually called *grouping*.

Heuristic based on Benders' decomposition. [*Sexton, Bodin* (1985a), *Sexton, Bodin* (1985b)] have solved a single-vehicle pickup and delivery problem, with desired delivery times. The developed heuristic uses Benders' decomposition applied to the mathematical formulation of PDPTW. The mathematical formulation is different from the formulation given in this paper, but it is also mixed integer non-linear program. Benders' decomposition separates the routing and scheduling components.

The scheduling component is the dual of the network flow problem whose structure allows exact solutions to be found quickly using a one-pass algorithm.

The *routing phase* is solved by a heuristic called *space-time heuristic*. The heuristic builds a route by considering capacity constraints, precedence constraints and space-time separations among stop locations. Space-time separation $\sigma_{i,j}$ between two stops i and j is defined by:

$$\sigma_{i,j} = d_{i,j} - t_i - t_j$$

where $d_{i,j}$ is direct travel time between stops, t_i is the latest allowable pickup time, if i is a pickup location, or desired delivery time, if i is a delivery location. The *latest allowable pickup time* is the difference between desired pickup time and direct travel time from pickup location to the corresponding delivery location.

An improvement procedure is also developed. It is based on repositioning of one stop location in the route. After each route change, the scheduling problem is solved again. The new schedule is compared with the best known so far. The algorithm will stop when no improvement can be achieved.

The authors have solved problems of up to 20 requests in a reasonable time.

The authors have applied the same algorithm as a part of solving the multi-vehicle problem. The procedure starts with a clustering step, followed by solving 1-PDPTW for each vehicle. Improvements were achieved by moving one request per time from one route to the other. Results on 85 requests and 7 vehicles were encouraging.

Mini-clustering/Routing construction method. The problem attacked by the mini-clustering idea [*Desrosiers, Dumas, Soumis* (1988)] is a large-scale (100-800 requests) multi-vehicle HTP. The set of vehicles is homogenous. The capacity of a vehicle is two-dimensional: wheelchairs and passengers. The starting and ending locations of vehicles are in one of several depots. Other characteristics: desired pickup *or* delivery time, departure time interval and arrival time interval, loading and unloading durations. The *piece of work* is defined as an interval in which the driver and the vehicle are available.

The main contributions of the research are:

- innovative concept of *mini-clusters*, where mini-cluster is defined as a period during which vehicle is non-empty
- generalization of the column-generation algorithm for the m-TSP with time windows [*Desrosiers, Soumis, Desrocher* (1984)]
- method of decomposition in time slices in order to handle very large problems

Algorithm consists of:

First stage : Mini-clustering of requests

- Build an initial solution to the problem by using an insertion heuristic developed by Roy et al. (1984)
- Reoptimize each of the routes by using exact dynamic programming algorithm [*Desrosiers, Dumas, Soumis* (1986)]
- Construct mini-clusters by dividing routes such that one mini-cluster is equal to the period within which a vehicle is non-empty. Note that each mini-cluster has to be served in a specific time-window.

Second stage : Mini-clusters assignment to vehicles

- Mini-clusters assignment to vehicles will be done in a way that results in a set of routes which minimizes the number of pieces of work and total travel time. The problem is m-TSPTW. The mathematical programming formulation is a set partitioning type problem which is solved by delayed column generation embedded into a branch and bound tree.

Third stage : Reoptimization

- Mini-clusters are reopened and each route is reoptimized by an using exact dynamic programming algorithm for solving 1-PDPTW [*Desrosiers, Dumas, Soumis* (1986)].

When the routing among mini-clusters is too large, the day is divided into successive overlapping time slices and a sequence of smaller problems is solved. In experiments, the authors have used 20-30 minutes long intervals each having 30-40 mini-clusters. The average number of requests in a mini-cluster is 4-10.

Tests have been done on instances with:

- 90-190 requests, 31 vehicles, resulting in 85 mini-clusters. Running time: 1-5 minutes
- 410 and 880 requests, 53 vehicles, resulting in 282 mini-clusters. Problem solved by the use of time slices. Running time: 6-20 minutes.

Improvement of about 10% have been reached comparing to Roy et al. (1984) insertion heuristic.

Since then, the authors have done further improvements in model and algorithm:

- Fleet of vehicles is expanded to be heterogeneous [*Dumas, Desrosiers, Soumis* (1989a)]
- In order to handle very large problems, a method of decomposition in both time slices and sub-areas were introduced [*Dumas, Desrosiers, Soumis* (1989b)]
- New parallel insertion heuristic is designed for mini-clustering [*Desrosiers, Dumas, Soumis, Taillefer, Villeneuve* (1991)]
- Delayed column generation approach is used for mini-clustering (Ioachim et al, 1995)

The final version of the algorithm can be shown by Figure 3 taken from [*Desrosiers, Dumas, Solomon, Soumis* (1995)].

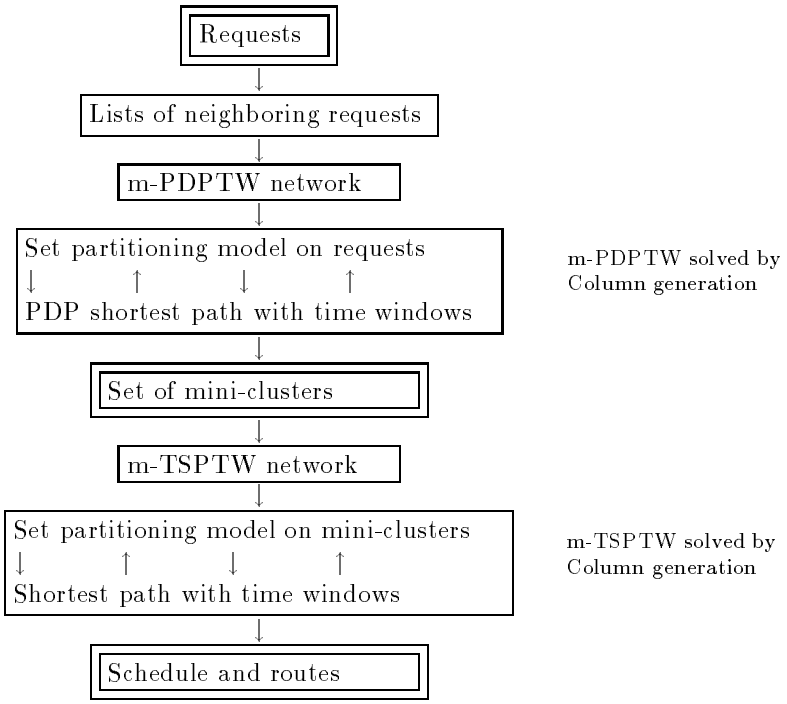


Figure 3: Mini-clustering/Routing construction method

Insertion heuristics

Insertion heuristics build a set of routes by inserting one request at a time into one route. Insertion procedures can be classified:

- sequential insertion procedure: generate one route per time, i.e. new route can be initialized when the current route is 'full',
- parallel insertion procedures: routes are constructed in parallel i.e. one request is scheduled to one active route.

Sequential insertion heuristic for PDPTW is first developed in connection with the development of real-time algorithms for dial-a-ride systems in Haddonfield, NJ, and Rochester, NY, in

the 1970s. The main concepts stated in that work are further developed and improved by other authors.

Sequential Insertion. One of the papers on insertion algorithms for PDPTW was published by [*Jaw, Odoni, Psaraftis, Wilson* (1986)]. The authors have developed heuristic algorithm for the m-DARPTW. The time windows are characterized by each customer specifying either desired pickup time or desired delivery time. Other characteristics of the problem are that the fleet of vehicle does not have to be homogenous, that a vehicle is not allowed to idle when carrying passengers, and that loading/unloading time exists and it can be different for each customer. A customer can also be rejected.

The *objective function* is a weighted sum of customers' dissatisfaction and system costs. Calculation of customer's dissatisfaction is based on excess riding time and deviation from desired pickup or desired delivery time. The objective function is defined such that during heavy-demand periods, more emphasis is placed on the system costs relative to service quality.

In order to serve customers better, the authors have introduced *service quality constraints*:

- no customer will be picked-up before desired pickup time and no customer will be delivered after desired delivery time i.e. hard one-sided time windows
- maximum riding time for each customer
- maximum deviation from desired pickup time or desired delivery time.

The given heuristic algorithm is a *sequential insertion procedure*, named ADARPTW. The algorithm starts by sorting customers in increasing order of their pickup times. ADARPTW inserts one customer at a time into one vehicle's schedule. The day is split into time slices of 20-30 minutes in length.

The main steps in the process of inserting a new request are:

Step 1: For each vehicle:

- search for all feasible insertions of new customer. For this purpose, a fast screening test is developed by introducing: (a) two-sided time windows (b) formulas for maximum amount of time by which every stop can be advanced or delayed in time without violating the time-window constraints.
- find the insertion that results in minimum increase of objective function value. The increase in the objective function value is called *additional cost*.

Step 2: If it is infeasible to insert customer in one of the routes, the customer will be rejected. Otherwise, the customer will be served by the vehicle whose additional cost is minimal.

Computationally, the randomly generated instances with four vehicles and 250 customers

were solved in 20 seconds. For the real-life instances the running time was around 12 minutes when the number of customers were 2600 and the fleet had 28 vehicles of three different types.

The restriction of the algorithm is that a new request can be inserted only within one schedule block.

This restriction and some others were improved in [*Madsen, Ravn, Rygaard* (1995)] while the authors were developing REBUS package to be used by Copenhagen Fire Fighter Services for the transportation of the elderly and handicapped. Additional problem characteristics that can be handled by REBUS are: multi-dimensional vehicle capacity and drivers' lunch breaks. The package also has the capability of allowing the operator to choose the objective function.

The preprocessing step of the algorithm is sorting requests by the difficulty to be serviced. Difficulty is measured on the basis of time windows, maximal travel time, and load. Also, each request insertion is ranked by a measure that includes: driving time, waiting time, deviation from desired service time, and capacity utilization.

REBUS is designed with the objective of applying in a dynamic environment for on-line scheduling. The performance was very satisfying: REBUS was 6-10 times faster than ADARPTW, and on instances with 300 requests, and 24 vehicles, running time was less than 10 seconds. A new request was scheduled in less than 1 sec.

The quality of the solutions was very similar to those of [*Jaw, Odoni, Psaraftis, Wilson* (1986)]

Parallel Insertion. [*Toth, Vigo* (1997)] have developed a parallel insertion procedure for solving HTP. Each user must be transported by a suitable vehicle from an origin to a destination location. Each customer specifies the desired service time and maximal riding time. The fleet of vehicles is nonhomogenous. The problem objective is to minimize use of taxis, service time violation, riding time violation, and routing costs.

The authors divide the city into 290 zones. The distance among zones is used to evaluate the travel times among stop locations. The time windows constraints were relaxed by the introduction of a linear inconvenience penalty in the objective function.
The algorithm has two phases:

Phase 1 = Routes initialization: determination of pivot requests for each of r routes. The procedure is determined by:

- number of *initial routes* r, which is evaluated as the minimum number of vehicles needed to serve 60% of all requests. (Percentage is determined on the basis of computational experiments.) In the calculation of r only capacity constraints are considered. (Example: Start with 100 requests. To serve 60 requests, with vehicles of capacity four, 15 vehicles are needed)
- *vehicles' efficiency measure*: vehicles are sorted in decreasing order of their efficiency. The efficiency of a vehicle is a measure based on the vehicle's capacity, cost, and loading time.
- requests' difficulty measure: requests are sorted in decreasing order of their difficulty. Difficulty measure includes: difficulty of serving corresponding customer, closeness to other requests, allowable excess time, and loading time.
- difficulty measure of inserting a request into a route: This is the sum of: (1) the request difficulty measure (2) the distance of the request from all other requests and (3) difficulty measures of inserting the request into route of more efficient vehicles
- linear inconvenience function

Phase 2 = Insertion of unscheduled requests: Done by solving a min-cost Rectangular Assignment Problem on the insertion cost matrix C. A row of C corresponds to an unscheduled request, and a column corresponds to a route. The element (i, j) of C is the additional cost (increase of objective function value) due to the cheapest insertion of a request i to route j.

If an unscheduled request cannot be inserted in any of the routes and if there are no vehicles available, the degree of difficulty of that trip is increased and the whole insertion procedure is restarted. This situation never happened in the process of solving of real-world instances.

The overall complexity of the algorithm is $O(kn^2)$.

Computational experience was reported for solving five real-life instances with about 300 requests each. The parallel insertion heuristic obtained substantial solution improvement compared to hand-made schedules: The number of vehicles used is reduced by almost 40% and the number of taxi rides used by more than 50%. The objective function value decreased by more than 35%. The average duration of service time violation is reduced by almost 70%, and the maximal service time violation is reduced by more than 85%. The running time was around 30 seconds.

The authors have done many experiments to tune the algorithm i.e. to get good values for a number of different parameters that the algorithm depends on. One undesirable characteristic

of the algorithm was that while solving large and strongly constrained instances, the heuristic tended to produce several short routes (typically served by taxis). In order to improve on that, the authors have developed a refining scheme - tabu threshold procedure [*Toth, Vigo* (1995), *Toth, Vigo* (1997)].

4.1.2 Tour improvement heuristics

Tour improvement procedures start with a set of feasible routes and seek to improve the solution through a sequence of steps. Almost all improvement heuristics for routing problems are actually some (simple or complex) version of a *local (neighborhood) search algorithm.*

Local search is based on what is perhaps the oldest optimization method - trial and error [*Papadimitrou, Steiglitz* (1982)]. Surprisingly, this method has proven quite successful on a variety of difficult combinatorial optimization problems.

A local search algorithm is built around a *neighborhood search procedure*, which given a solution, examines all solutions which are closely related to it and finds a best such neighbor (if one exists). The definition of 'closely related' is based on the definition of 'neighborhood', which varies from one local search algorithm to the other. The particularly successful heuristics for single-vehicle routing problems are based on 'neighbors' of a route obtained from it by doing a limited number of interchanges of route edges with non-route edges.

Local search methods, for combinatorial optimization problems, vary from simple algorithms [*Papadimitrou, Steiglitz* (1982)] to very complex modern heuristic like simulated annealing and tabu search [*Gendreau, Laporte, Potvin* (1997)]. This section will explain classic insertion algorithms while the following section will give an overview of modern heuristics.

Cyclic Transfers. Improvement procedure described in [*Thompson, Psaraftis* (1993)] is an application of cyclic transfer. Cyclic transfer is a neighborhood search developed by Thompson in 1988 and was used to solve to multi-vehicle routing and scheduling problems.

The main idea behind the *cyclic transfer* application to the pickup and delivery problems is in the effort to improve a schedule by moving some requests among routes. Transfer is called cyclic because of the concept: for a given variation of routes, say (1,5,8), a certain number of requests will be moved from route 1 to route 5, a certain number of requests will be moved from route 5 to route 8 and, at the end, a certain number of requests will be moved from route 8 to route 1. A special class of cyclic transfers is *cyclic k-transfers* where k is the number of requests that will be moved. Another special class is *b-cyclic k-transfer* where b defines the number of routes among which requests will be transferred i.e. the class of the route variation. A generalization of cyclic transfers can be done by allowing a certain number of dummy requests in each (or some) of the routes.

The *cyclic transfer neighborhood* of a feasible solution r is the set of feasible solutions reachable from r via a cyclic transfer. A feasible solution r is a *cyclic transfer optimal* if no member of cyclic transfer neighborhood of r has a better objective function value.

Thompson and Orlin developed in 1989 a general methodology for the cyclic transfer neighborhood search which involves transforming the search for negative cost cyclic transfers into a search for negative cost cycles on an auxiliary graph. Unfortunately, the construction of the auxiliary graph is an NP-hard problem. Arc cost c_{ij} of the auxiliary graph is the increase in objective function due to simultaneously removing node i and inserting node j in one route. Thus the calculation of arc costs is a TSP type of problem on a new set of nodes.

The algorithm was applied on a variety of routing problems. The worst-case analysis has shown arbitrarily bad performance on a majority of them, even on the multi-depot Euclidean vehicle routing problem. Luckily, the structure of the worst-case problem instances is not typical in practical problems. Experimental results on real-life problems have exhibited behavior similar to Lin-Kernighan k-opt TSP heuristic which also has unbounded arbitrarily worst-case performance but perform quite well in practice.

Because of these and some other difficulties, the authors propose:

- polynomial-time approximation function for the arc cost calculation,
- restriction of the cyclic transfer neighborhood by:
  - limiting initial search to 2-cycles, 3-cycles, or
  - generating and searching only part of the auxiliary graph.

Fortunately, these ideas reduce the computational requirements to a reasonable level.

Computationally, the experiments were done on ship routing and scheduling instances given by Psaraftis et al. in 1985. The objective was to minimize lateness. Initial solutions were constructed by the Psaraftis et al. (1984) heuristic. Those solutions were improved by 60-100% in less than two minutes (a 100% improvement means there is no location stop served late). The number of vehicles is up to 10 and the number of demands is up to 50.

Variable Depth Search. [*Van Der Bruggen, Lenstra, Schuur* (1993)] have developed a local search improvement procedure based on the variable-depth search, similar to the Lin-Kernighan algorithm for the TSP. The search is built upon basic types of arc-exchange designed by the authors. The procedure has been tested on the single-vehicle pickup and delivery problem with time windows.

A traditional *k-exchange* mechanism for TSP is an exchange of k arcs in the route by some other k arcs. The route that cannot be improved by k-exchange is k-optimal. The most used are 2-exchanges, 3-exchanges and Or-exchanges. Or-exchange is a 3-exchange in which only strings of one, two, or three consecutive vertices are relocated between two other vertices. For 2-exchange and Or-exchange, Savelsbergh has developed effective techniques for checking feasibility and profitability in the case of PDPTW.

A more sophisticated technique is the Lin-Kernighan variable-depth search where the number of arcs to be replaced is determined dynamically. [*Van Der Bruggen, Lenstra, Schuur* (1993)]

have used this idea and the techniques developed by [*Savelsbergh* (1990)] to introduce a new algorithm combining seven arc-exchange techniques.

The authors have developed their own construction algorithm based on an improvement procedure. A random initial solution, possibly infeasible, is improved until it becomes feasible. A new objective function measures the infeasibility in time constraints, which are the only constraints that could be violated.

The authors have tested their algorithm on real-life instances as well as on randomly generated ones. On real-life problems with 5 to 38 customers, the algorithm has produced solutions that were either optimal (more than 50% of instances) or within 0.8% of optimal. The time window constraints were relatively tight. The running time was less than a minute. On generated instances with 100 customers and wide time windows the algorithm was much slower. The initial solution was bad and the running time of the improvement procedure was more than 10 minutes. The problem was that when time windows are wide the set of feasible solutions is much bigger allowing the construction of a poor initial feasible solution.

4.2 Modern heuristic methods

4.2.1 Metaheuristics

Metaheuristics were designed in the early 1980s in order to attack complex and difficult combinatorial optimization problems that arise in many practical areas. A thorough bibliography is published by [*Osman, Laporte* (1996)]. A metaheuristic is an iterative generation process which guides some subordinate heuristic. Guidance is performed by combining intelligently different concepts for exploring and exploiting the search space. Metaheuristics include, but are not limited to: simulated annealing, tabu search, threshold algorithms, neural networks, genetic algorithms, ...

Simulated Annealing. In order to overcome the possibility of an algorithm getting stuck in a poor local optimum the authors of variable-depth search have developed a simulated annealing algorithm [*Van Der Bruggen, Lenstra, Schuur* (1993)].

Simulated annealing (taken from [*Lawler, Lenstra, Rinnoy Kan, Shmoys* (1985)]) was developed by Kirckpatrick, Gelatt, Vecchi in the early 1980s when the authors postulate that there are certain similarities between combinatorial optimization problems and large physical systems studied in statistical mechanics. *Statistical mechanics* deals with analyzing aggregate properties and energy levels of a large number of atoms in liquids and solids at different temperatures. Experiments that reveal the low temperature state of a material are performed by a process referred to as *annealing*. The material under study is first melted and then the temperature is slowly lowered with a long time spent at temperatures near freezing point. The period of time spent at each temperature must be sufficiently long to allow a thermal equilibrium to be achieved. This annealing process can be simulated by using the following procedure: given a

configuration of the elements of the system, randomly displace the elements, one at a time, by a small amount and calculate the resulting change in the energy. If energy change is negative then accept the displacement and use the resulting configuration as the starting point for the next iteration. If energy change is nonnegative, then the displacement is accepted with a probability that depends on the energy change and temperature. Repetition of this step continues until equilibrium is achieved. At this point, the temperature is lowered and the procedure is repeated.

The simulated annealing can be implemented to solve vehicle routing problems as follows: at each iteration (temperature level) each better, randomly generated, feasible solution is accepted, and every worse feasible solution is accepted with some probability, which depends on the temperature i.e. on the number of applied iterations. A mechanism for accepting worse feasible solutions is a mechanism against getting stuck in local optimums.

In addition, any local search procedure can be made more flexible by allowing moves to infeasible solutions. To record and control badness of feasible solutions, the objective functions can be augmented with the term that punishes the amount of violation of the restrictions. Simulated annealing is a good environment for implementing this idea in a dynamic way. Such a method is known as the *penalized simulated annealing.*

[*Van Der Bruggen, Lenstra, Schuur* (1993)] have applied penalized simulated annealing to PDPTW for generating initial feasible solutions. The results were very good even when time windows were wide. The one disadvantage was the slowness of the procedure.

Tabu Threshold. The motive for developing yet another improvement procedure was the fact that the parallel insertion heuristic proposed by [*Toth, Vigo* (1997)] tends to produce a number of undesirable short routes. So [*Toth, Vigo* (1997)] extended their parallel insertion algorithm for solving HTP to a metaheuristic, by developing a local search improvement procedure. The improvement procedure can be described as restrictive neighborhoods and simple descent procedures. More precisely, the procedure is Tabu Treshold proposed by Glover in 1995. Tabu Treshold (TT) is a variation of tabu search procedure, effective for large-scale instances of optimization problems when limited computer time is available.

Tabu search is a modern local search technique which examines a neighborhood of a current solution in order to make the next move to the 'best' neighbor. To avoid cycling, solutions that were recently examined are forbidden, or *tabu* for a number of iterations. There are lots of variations among tabu search techniques, because the number of parameters that define technique can be tuned differently, and because the 'best neighbor' can have a completely different meaning, sometimes even allowing moves to infeasible solution.

The main innovation in the *tabu threshold* (TT) method is a *candidate list strategy*: the neighborhood of the current solution is subdivided into subsets of moves. At each iteration of the TT procedure, one of the subsets is chosen and the best admissible move belonging to the subset, if any, is performed. As a consequence, the number of moves considered is reduced, and

the overall computational effort is decreased.

TT is based on the alternation of two steps: improve phase and mixed phase. Improve phase is used to get to the local optimum, and mixed phase is used to try to escape local optimum. In the *improve phase* one subset of moves is examined, at each iteration, and the best move which improves the current best solution is executed. The order in which subsets are examined is changed after each complete examination of the neighborhood. When the last examination of the complete neighborhood improve the solution for less than 0.5%, the improve phase will terminate. In the *mixed phase* a move to a non-improving solution is allowed. Duration of mixed phase is bounded by a random parameter chosen from the interval [2,4].

Solution *cycling* is prevented by changing the order in which subsets are examined, rather than using the memory structure as in Tabu Search.

The *neighborhood* is the set of intra-route and inter-route movements of requests. The *neighborhood subset* S_i, for each request i, contains all moves relative to request i. The movements are:

> **Request insertion:** moving request i to the best position of (a possibly different) route.
>
> **Trip exchange:** swap request i with a request from some other route. Insert requests in the best positions.
>
> **Trip double insertion:** moving request i from route j_1 to route j_2, then moving request k from route j_3 to route j_1. Insert requests in the best positions.

The time required for one subset examination is $O(nk^3)$, where n is the number of requests, and k is the average number of requests served by one route. Computational experience has shown that the tabu threshold algorithm is time consuming. To get reasonable running times some tests were done by using only the improve phase of the algorithm:

> **(A)** Improve phase of TT has improved the solution constructed by the parallel insertion algorithm for about 10% in each of the following characteristics: number of taxis used, travel times and distances, value of objective function. The running time was around 5 minutes.
>
> **(B)** Whole TT procedure has improved solution, constructed by 'parallel insertion algorithm + TT improvement phase', by more than 5% in almost all characteristics. Running time was almost an hour.

Constraint-Directed Search. One of the few papers on metaheuristic algorithms developed for pickup and delivery problems with time windows is [*Potvin, Rousseau* (1992)]. The problem addressed was the multi-vehicle dial-a-ride problem with service quality constraints.

Service quality constraints are:

- ride time will not exceed some prespecified threshold

- pickup time cannot be earlier than the desired pickup time, and delivery time cannot be later than the desired delivery time
- pickup or delivery time will not deviate from the desired time by more than the pre-specified amount
- no vehicle can stay idle while there are customers on board

Time windows are defined from service quality constraints.

Application of the constraint-directed search method to the problem has 3 phases:

(1) Initialization phase (optional)
(2) Constraint-directed search - beam search phase
(3) Post-optimization phase - exchange heuristics

Initialization phase is set in a way to include around 20% of the customers in initial routes. The initialization phase starts with clustering customers based on spatial and temporal measures. Two measures were used:

- maximum riding time between two locations: bounded by some prespecified value
- measure of zigzagging in serving two customers in one route: $d_{+1,+2} + d_{+2,-1} \leq \alpha d_{+1,-1}$. Parameter α is set to 1.4

The customers are sequenced within each cluster by building feasible routes.

Beam search phase represents embedding the [*Jaw, Odoni, Psaraftis, Wilson* (1986)] insertion heuristic within constraint-directed beam search. One iteration of the algorithm inserts one customer in a route:

basic mechanism :

- sort non-scheduled customers in decreasing order of 'earliest pickup time' (EPT)
- choose several customers to be inserted (search operator 1)
- for every chosen customer: find all feasible insertion points (search operator 2). Only valid insertion is within one schedule block (the same restriction as in [*Jaw, Odoni, Psaraftis, Wilson* (1986)]).
- the W best new partial routes are selected by *state evaluation*. Those routes are used for further expansion and the remaining solutions are discarded. Parameter W is known as the *width of beam search*.

state evaluation :

- *state evaluation* measures the degree of satisfaction of stated preferences (for the definition of stated preference look into the next paragraph).

Post-optimization phase has three distinct improvements:

swap-1 - eliminate smaller routes by inserting their customers in longer ones

swap-2 - exchange positions of stops of two distinct customers

optimum - finds the optimum sequence of customers within a schedule block. Applied to schedule blocks with three or less customers.

Hard constraints are the vehicle capacity constraints, vehicle availability constraints, maximum deviation from desired pickup or delivery time, and maximum riding time. *Soft constraints i.e. stated preferences* are: deviation from desired pickup or delivery time, deviation from desired riding time, and operation costs which includes the number of vehicles, total vehicle time, average utilization of a vehicle, and distribution of customers among vehicles. Soft constraints are called *stated preferences* in artificial intelligence literature. The model uses a quadratic utility function for each preference.

This algorithm really looks very much like [*Jaw, Odoni, Psaraftis, Wilson* (1986)] where stated preferences are modeled by weighted objective function. The only strong difference is a consideration of several customers in parallel and choosing several partial solutions to continue with.

Computational results, in the paper, are given for instances with 90 customers. The algorithm was tested for a different set of control parameters: beam search width W, number of customers whose insertion will be considered in one iteration (search operator 1), number of routes in which an insertion will be tried. For (2,2,2) values of control parameters, the algorithm was faster but results were worse than in (3,3,3) choice for which the algorithm was, as expected, slower. Compared to [*Jaw, Odoni, Psaraftis, Wilson* (1986)] the solutions were 5-15% better but the algorithm needed 2-5 times more time (125-350 sec).

Automated Learning Techniques. One more application of artificial intelligence methods to pickup and delivery problem is reported in [*Shen, Potvin, Rousseau, Roy* (1995)]. The authors have described an expert consulting system for a dispatcher working in a courier service company. The problems in the company are dynamic, requiring real-time decision making. The dispatcher has, upon receiving a new request, to assign the request to one of the existing routes (vehicles). The objective is to minimize operational costs and maximize customers' satisfaction.

The computer system is based on automated learning techniques. The system consists of two parts: the dispatching module and the learning module.

The main functions of *the dispatching module* are:

- Request localization: Translating the address into an element of network structure.
- Travel times evaluation: Travel distance between two points is estimated on the basis of a stored distance matrix for 502 zones of the city of Montreal. Estimation of travel time includes consideration of the average travel time (as a function of distance), meteorological conditions, driver's productivity, and traffic congestions.
- Allowing route selection that will serve new request: The system presents nine characteristics of new proposed routes allowing and helping a dispatcher to make a final decision.

The learning module is aimed at suggesting the best routes in which to insert the new request. The module is learning by example. This is more convenient, because it is easier to get decision examples from dispatchers than to extract explicit decision rules. The learning is based on a backpropagation neural network. The backpropagation neural network is typically composed of three layers: input layer, hidden layer and output layer. Each layer is connected to the next layer via weighted connections. The power of the backpropagation network is in its ability to adjust weights on its connections to learn specific tasks. Neural network learning process is called training. A large number of examples of the dispatcher's decisions are provided to the network during the training phase. A trained neural network can assist the dispatcher by giving him a choice from a set of ranked routes. This allows a dispatcher to concentrate on a small subset of 'good' routes in which a new request should be inserted.

In the experiments on real data, the dispatcher selected 80% of routes that were ranked first, second or third by the neural network. Moreover, 94% of the chosen routes were ranked as first four by neural network. These results indicate that the developed system can be truly used to filter interesting alternatives and to present only them to the dispatcher.

4.3 Heuristics with finite bounds

The only heuristics with finite bounds, as per authors' knowledge, are the algorithms for solving single-vehicle pickup and delivery problem without time windows and capacity constraints (1-PDP). This is because by introducing time windows suitably, any instance can be made arbitrarily bad. The heuristics with finite bounds are based on the algorithms for solving TSP.

We will describe here one of these algorithms developed by Psaraftis (1983). The problem is the single-vehicle immediate-request dial-a-ride problem, where the distances are Euclidean.
Algorithm:

Step 1: Construct traveling salesman tour through all stop locations.

Step 2: Choose any stop location as a starting point. Traverse TSP tour clockwise and do not: (a) visit any delivery location before its corresponding pickup location, and (b) visit already visited stop location.

Step 3 (optional): Apply improvement procedure: sequence of local interchanges.

Step 4 (optional): Repeat Step 2 (and 3) traversing TSP tour counterclockwise.

Step 5 (optional): Repeat Step 2 (3, and 4) choosing different stop locations for starting points.

The tour with minimum length is the solution.

Algorithm complexity and worst-case performance depend on the algorithm used in Step 1 for a traveling salesman tour construction. If the minimum spanning tree heuristic (Prim, 1957) is used, the algorithm complexity is $O(n^2)$, and the length of the constructed tour is at most four times longer than the optimal. If the Christofides (1976) heuristic is used, it improves the performance ratio to three times the optimal length, but the running time increases to $O(n^3)$.

The biggest problem solved, by using Prim's TSP heuristic, was a problem instance with 50 customers.

5 Dynamic PDPTW

Despite the brevity of the material in the area of the dynamic PDPTW, the topic is important because of the trends in traffic and transportation nowadays. The developments in telecommunication, information, computer and transportation systems are extensive, thus bringing real-life traffic data and transportation conditions into the dispatcher center. Researchers need to develop the methods and techniques for solving such real-life and real-time problems. The new methods have to benefit from the flow of incoming information.

Dynamic problem is a problem about which not all information is known at the time when the execution of a partial solution has to start. Dynamic problem is also called a *real-time problem.*

A good example of the dynamic pickup and delivery problem is the CCPDP. Inflow of requests is coming in real-time. The known requests have to be served before all requests for the day are received. The status of the system changes constantly. The decisions have to be made quickly. This kind of system is also referred as *demand responsive system.*

Some of the methods described in this paper have been reported as being used in dynamic environments i.e. for scheduling incoming calls. For example [*Madsen, Ravn, Rygaard* (1995)] describe the DARP system REBUS. The majority of these methods applied to dynamic problems were static algorithms that run on demand, i.e. when a new request has to be scheduled. This approach to solve a dynamic problem is called the *dynamic application of a static model* [*Desrosiers, Dumas, Solomon, Soumis* (1995)] and consists of repeatedly solving a static problem.

It may be noted that the best methods that can directly fit into a dynamic environment are the insertion methods. The better a method is and the less improvement it requires, the better it could be used in real-time decision making.

In the only paper that deals with the real dynamic PDPTW [*Shen, Potvin, Rousseau, Roy* (1995)], the authors develop an adaptive decision support system for solving the true CCPDP. The method is automatic learning based on neural networks. It was successfully applied in practice as a dispatcher decision support system.

Two additional papers we review on dynamic vehicle routing problems are [*Psaraftis* (1988), *Psaraftis* (1995)]. In the first, the author describes the *rolling horizon principle* for dealing with dynamic problems. The solved problem is a routing of cargo ships in an emergency situation.

An algorithm, based on the rolling horizon principle, for solving PDPTW is:

Step 1: Calculate routes that will cover known requests in the next four hours. This is a *tentative assignment.*

Step 2: Select requests that have to be served in the next two hours. Assign them

to vehicles. This is a *permanent assignment.*

Step 3: Wait one hour. Go to Step 1.

In other words, we are looking four hours ahead and we fix routes in the next two hours. We repeat this every hour.

The other paper on dynamic vehicle routing [*Psaraftis* (1995)] gives examples of dynamic problems in practice, and lists the technology advances that can help a dispatcher decision support system. The author describes the current status of research in the area and gives directions on areas that require further development. A number of problems are still open: definitions, worst-case analysis, algorithms that will work well in dynamic problem versions, parallelization and distributions of algorithms... For further details and other practical dynamic vehicle routing problems see [*Psaraftis* (1995)].

6 Conclusions

The research of time constrained pickup and delivery problems emerged in the last 15-20 years. Researchers have developed a variety of heuristics and optimization methods.

The development of ***optimization methods*** started in the early 1980s and lasted almost a decade.

Among optimization methods the Dantzig-Wolfe decomposition/Column generation method has shown the best results, solving a special class of m-PDPTW with up to 50 requests. The dynamic programming algorithms for 1-PDPTW have solved problems of size 40.

Keeping in mind the size of real-world problems, the future of optimization methods in solving practical pickup and delivery problems is not promising. Fortunately, the optimization methods could be used for solving subproblems in one subarea if the problem can be partitioned. It can also be used as a precomputation step in a dynamic problem solving i.e. as a step for finding initial routes over the subset of requests known in advance.

Heuristics for solving real-life pickup and delivery problems began to appear in the literature in the 1970s. The performance of heuristics is difficult to compare. The trend has usually been to compare heuristic solutions with hand-made solutions, or with the solutions of some other heuristic on which the new heuristic is based on. The comparison with hand-made solutions looks inadequate, but it is a good base for proving that optimization using computers can do a good job. This is crucial for initiation and support of new research and development in the area.

Another possible way to compare heuristics is in checking the problem size a heuristic can solve and the amount of computer time and space it needs. Unfortunately, this comparison is unfair, considering the pace of computer hardware development and the diversity of computers available to different research groups.

Yet another difficulty, in comparing heuristic performances, lies in the fact that the majority of reported research originates from work on problems in practice. Clearly the diversity of practical problems is immense. Consequently, there are not too many papers on the same problem. Constraints can be different, understanding of customers' satisfaction can be different, objective functions can be different... This makes it hard to compare the underlying methods.

Anyway, the development of heuristics has reached the stage where large-scale practical problems, with thousands of requests, can be solved.

The best results among *construction heuristics* are reached by mini-clustering/column generation, the method based on mathematical programming. Instances of upto 880 requests with 53 vehicles can be solved with this method.

The efficiency of *insertion heuristics* depends very much on the measures of closeness, difficulty, satisfaction, evaluation of travel times... The authors have captured a great deal of detail in their models. Many real-life applications for DARP and HTP are based on these heuristics and work well in practice. In addition, these approaches have a bright future, because they can be applied directly to real-life dynamic problems.

Improvement heuristics are primarily based on arc-exchange procedures, and can be used in practice as improvements for both the dynamic and static case of a problem.

The development of *metaheuristic* methods for pickup and delivery problems has emerged in the last few years. Tabu threshold, simulated annealing, neural networks, and constraint-directed search, are among the applied approaches. All of these techniques could be used in solving real-life pickup and delivery problems. These techniques show that an interdisciplinary approach can help in solving some hard problems in this area.

The majority of published work on pickup and delivery problems with time windows (PDPTW) is on dial-a-ride problems (DARP). In the last several years, research in the handicapped persons transportation (HTP) has also emerged. In contrast to this, very little work has been done on pickup and delivery of packages and goods. Few papers report the work done on the courier company pickup and delivery problem (CCPDP).

Direct implementation of methods, for solving DARP or HTP is not a solution for CCPDP. The CCPDP is mostly uncapacited, and the time windows are wider. The constraint of the maximum riding time does not apply to CCPDP. The same is true with the constraint of 'not being allowed to wait while passengers are in the vehicle'. In addition, mini-clustering is not applicable to CCPDP because there is very little possibility of a vehicle being empty at any time during a day. All of those differences seem to imply that the set of feasible solutions is larger in CCPDP than in the problems where people are transported. Consequently, the methods for DARP and HTP, applied to CCPDP, will require much more time. Yet another difference is that CCPDP is a truly dynamic problem. The constant inflow of requests leaves little time for making decisions (and little time for the algorithm to give solutions).

It is also worth mentioning that many of the developed methods are based on the ideas used in solving vehicle routing problems (VRP) and/or traveling salesman problems (TSP). Keeping in mind that, in practice, VRPs are mostly static problems, and PDPs are mostly dynamic problems, the author feels that there is scope for new research and new discoveries.

One more fact in favor of future research in PDPTW is the pace of developments in telecommunications, computer science, information technologies, vehicle equipment industries... These theoretical, software and hardware developments bring new tools in the quest for methods to solve dynamic transporatation problems.

References

[*Assad* (1988)] Assad A.A. (1988) "Modeling and implementation issues in vehicle routing", in Vehicle Routing: Methods and studies, edited by Golden, B.L., Assad, A.A.

[*Bodin, Golden, Assad, Ball* (1983)] Bodin L., Golden B.M., Assad A., Ball M. (1983) "Routing and scheduling of vehicles and crews: the state of the art", Computers and Operations Research, 11(2), 63-211, 1983

[*Chvatal* (1983)] Chvatal V. (1983) "Linear Programming", W.H.Freeman and Company

[*Desrosiers, Dumas, Solomon, Soumis* (1995)] Desrosiers J., Dumas Y., Solomon M.M., Soumis F. (1995), "Time constrained routing and scheduling", in: Ball M.O., Magnanti T.L., Monma C.L., Nemhauser G.L., (eds.) Network Routing, Handbooks in Operations Research and Management Science, Volume 8, INFORMS, Elsevier Science, 141-284

[*Desrosiers, Dumas, Soumis* (1986)] Desrosiers J., Dumas Y., Soumis F. (1986), "A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows", American Journal of Mathematical and Management Sciences 6, 301-325

[*Desrosiers, Dumas, Soumis* (1988)] Desrosiers J., Dumas Y., Soumis F. (1988) "The multiple vehicle dial-a-ride problem", in Daduna J.R., Wren A. (eds.) Computer Aided Transit Scheduling, Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems 308, pp. 15-27, Springer Verlag 1988

[*Desrosiers, Dumas, Soumis, Taillefer, Villeneuve* (1991)] Desrosiers J., Dumas Y., Soumis F., Taillefer S., Villeneuve D. (1991) "An algorithm for mini-clustering in handicapped transport", Report G-91-02, Ecole des Hautes Etudes Commerciales, Montreal

[*Desrosiers, Soumis, Desrocher* (1984)] Desrosiers J., Soumis F., Desrocher M. (1984), "Routing with time windows by column generation", Networks 14, 545-565

[*Dumas, Desrosiers, Soumis* (1989a)] Dumas Y., Desrosiers J., Soumis F. (1989b) "Large-scale multi-vehicle dial-a-ride problems", Les Cashier du GERAD G-89-30, September 1989

[*Dumas, Desrosiers, Soumis* (1989b)] Dumas Y., Desrosiers J., Soumis F. (1989c) "The pickup and delivery problem with time-windows", Report G-89-17, Ecole des Hautes Etudes Commerciales (Ecole des HEC), Montreal, Canada

[*Dumas, Desrosiers, Soumis* (1991)] Dumas Y., Desrosiers J., Soumis F. (1991) "The pickup and delivery problem with time windows", European Journal of Operational Research 54, 7-22

[*Fisher* (1995)] Fisher M. (1995), "Vehicle routing", in: Ball M.O., Magnanti T.L., Monma C.L., Nemhauser G.L., (eds.) Network Routing, Handbooks in Operations Research and Management Science, Volume 8, INFORMS, Elsevier Science, 141-284

[*Gendreau, Laporte, Potvin* (1997)] Gendreau M., Laporte G., Potvin J.Y. (1997), "Vehicle routing: modern heuristics", in: Aarts E.,Lenstra J.K. (eds.) Local Search in Combinatorial Optimization, John Wiley and Sons, 311-336

[*Laporte* (1992)] Laporte G., (1992), "The vehicle routing problem: an overview of exact and approximate algorithms", European Journal of Operational Research 59, 345-358

[*Lawler, Lenstra, Rinnoy Kan, Shmoys* (1985)] Lawler E.L., Lenstra J.K., Rinnoy Kan A.H.G., Shmoys D.B. (eds.) (1985) "The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization", John-Wiley and Sons

[*Jaw, Odoni, Psaraftis, Wilson* (1986)] Jaw J.J., Odoni A.R., Psaraftis H.N., Wilson N.H.M. (1986) "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows", Transportation Research B, Vol.20B, No.3, 243-257, 1983

[*Madsen, Ravn, Rygaard* (1995)] Madsen O.B.G., Ravn H.F., Rygaard J.M. (1995), "A heuristic algorithm for a dial-a-ride routing and scheduling problem with time windows, multiple capacities, and multiple objectives", Annals of Operations Research 60, 193-208

[*Osman, Laporte* (1996)] Osman I.H., Laporte G. (1996) "Metaheuristics: a bibliography", in Metaheuristics in Combinatorial Optimization, Osman I.H., Laporte G. (eds.), Annals of Operations Research, Vol.63, Baltzer Science Publishers

[*Papadimitrou, Steiglitz* (1982)] Papadimitrou C.H., Steiglitz K. (1982) "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall

[*Potvin, Rousseau* (1992)] Potvin J.Y., Rousseau J.M. (1992) "Constraint-directed search for the advance request dial-a-ride problem with service quality constraints", Computer Science and Operations Research: New Developments in their Interfaces, O. Balci, R. Shrada, S.A. Zenios (Eds.), Pergamon Press, Oxfrod, England, 457-474

[*Powell* (1991)] Powell W.B. (1991) "Optimization models and algorithms: an emerging technology for the motor carrier industry", IEEE Transactions on Vehicular Technology 40(1)

[*Powell, Jaillet, Odoni* (1995)] Powell W.B., Jaillet P., Odoni A. (1995), "Stochastic and dynamic networks and routing", in: Ball M.O., Magnanti T.L., Monma C.L., Nemhauser G.L., (eds.) Network Routing, Handbooks in Operations Research and Management Science, Volume 8, INFORMS, Elsevier Science, 141-284

[*Psaraftis* (1980)] Psaraftis H.N. (1980) "A dynamic programming solution to single vehicle many-to-many immediate request dial-a-ride problem", Transportation Science, Vol.14, No.2, May 1980

[*Psaraftis* (1983a)] Psaraftis H.N. (1983a) "An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows", Transportation Science, Vol.17, No.3, August 1983

[*Psaraftis* (1983b)] Psaraftis H.N. (1983b) "Analysis of an O(n^2) heuristic for the single vehicle many-to-many euclidean dial-a-ride problem", Transportation Research B, Vol.17B, No.2, 1983

[*Psaraftis* (1988)] Psaraftis H.N. (1988) "Dynamic vehicle routing problems", in Vehicle Routing: Methods and studies, edited by Golden, B.L., Assad, A.A., 223-248

[*Psaraftis* (1995)] Psaraftis H.N. (1995) "Dynamic vehicle routing: status and prospects", Annals of Operations Research 61, 143-164

[*Savelsbergh* (1990)] Savelsbergh M.W.P (1990) "An efficient implementation of local search algorithms for constrained routing problems", European Journal of Operational Research 47, 75-85

[*Savelsbergh, Sol* (1995)] Savelsbergh M.W.P, Sol M. (1995) "The general pickup and delivery problem", Transportation Science 29(1), 17-29

[*Shen, Potvin, Roy, Rousseau* (1993)] Shen Y., Potvin J.Y., Roy S., Rousseau J.M. (1993) "Integrating operations research and artificial intelligence techniques for vehicle dispatching", Technical Report CRT-939

[*Shen, Potvin, Rousseau, Roy* (1995)] Shen Y., Potvin J.Y., Rousseau J.M., Roy S. (1995) "A computer assistant for vehicle dispatching with learning capabilities", Annals of Operations Research 61, 189-211

[*Sexton, Bodin* (1985a)] Sexton T.R., Bodin L. (1985a) "Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling", Transportation Science 19, 378-410

[*Sexton, Bodin* (1985b)] Sexton T.R., Bodin L. (1985b) "Optimizing single vehicle many-to-many operations with desired delivery times: I. Routing", Transportation Science 19, 411-435

[*Solomon, Desrosiers* (1988b)] Solomon M.M, Desrosiers J. (1988) "Time window constrained routing and scheduling problems", Transportation Science 22 (1), 1-13

[*Thompson, Psaraftis* (1993)] Thompson P.M., Psaraftis H.N. (1993) "Cyclic transfer algorithms for multivehicle routing and scheduling problems", Operations Research, Vol. 41, No. 5, September-October 1993

[*Toth, Vigo* (1995)] Toth P., Vigo D. (1995), "Fast local search algorithms for the handicapped persons transportation problem", Technical Report DEIS-OR-95-1(R) (revised version) to appear in J.P.Kelly, I.H.Osman (eds), "Meta-Heuristics: Theory and Applications", Kluwer 1996.

[*Toth, Vigo* (1997)] Toth P., Vigo D. (1997), "Heuristic algorithms for the handicapped persons transportation problem", Transportation Science, Vol. 31, No. 1, February 1997

[*Van Der Bruggen, Lenstra, Schuur* (1993)] Van Der Bruggen L.J.J., Lenstra J.K., Schuur P.C. (1993) "Variable-depth search for the single-vehicle pickup and delivery problem with time windows", Transportation Science, Vol. 27, No.3, 298-311